Interface

2003 July

Contents 2003 July

［表紙デザイン：㈱プランニング・ロケッツ］

特集

高速ロジックインターフェースからPCI Express，10Gigabit Ethernetまで

Interface

Contents 2003 July

話題のテクノロジ解説

ショウレポート&コラム

一般解説&連載

情報のページ

連載「フリーソフトウェア徹底活用講座」,「やり直しのための信号数学」は,お休みさせていただきます.

先端要素技術の総合展示会

TECHNO-FRONTIER 2003

北村俊之

「次の時代が見えてくる」をキーワードに，エレクトロニクス，メカトロニクスの先端要素技術，製品を一堂に展示する「TECHNO-FRONTIER 2003」が4月16日（水）～18（金）の3日間，日本コンベンションセンターで開催された．主催は（社）日本能率協会．本展示会では「第21回 モータ技術展」，「第18回 電源システム展」，「第16回 EMC・ノイズ対策技術展」，「第12回 モーション・エンジニアリング展」，「第12回 ボード・コンピュータ展」，「第5回 熱対策技術展」，「第4回 高効率・省エネ促進技術展」，「第3回 カーエレクトロニクスデバイス展」，「第3回 Bluetooth Expo」の九つの展示会と関連の七つのシンポジウムが同時に開催された．また，特別企画として「開発・設計を支援する解析技術」，「リニアモータと応用技術」，「産・学技術移転（TLO）交流プラザ」，関連企画として「海外部品調達展2003」も併せて開催されるなど，大規模な展示会となっていた．最終的な来場者数は，3日間で122,940人だった．今回は，「第12回 ボード・コンピュータ展」および「第3回 Bluetooth Expo」を中心にレポートする．

● 第12回 ボード・コンピュータ展

ボード・コンピュータ展は，組み込み機器，システムの最適構築を支援する各種バスボードからソフトウェア，開発支援ツール，周辺機器の実用技術の展示会となっている．

アドバネットは，コンパクトPCIバス，VMEバス，PCIバス，PMC，StarFabricブリッジボードなど，多様な規格に対応したボードを多数展示していた．なかでも今回注目を集めていたのが，Mobile Pentium4-Mを搭載したコンパクトPCIバス対応のCPUボード「A6pci8012」（**写真1**）．こちらは次世代産業機器に欠かせないCPUパワーを提供する製品であるという．また，同ブースではStarFabricブリッジボートの各種新製品も展示されており，こちらも最近注目を集めているとのことだった．

〔写真1〕アドバネットのA6pci8012

このStarFabric規格を積極的に紹介したロッキーは，StarGateコンパクトPCIボードの新製品を展示していた（**写真2**）．同製品は2ポートのバンドリングにより，同時に送受信5Gbpsを実現している．また従来のBIOSやOSのPCIソフトウェアと完全互換であるため，PCIバス拡張（64ビット/66MHz）やマルチプロセッサシステムを容易に構築できるという．

〔写真2〕ロッキーのStar Gateコンパクト PCIボード

ソリトンシステムズでは，300Mバイト/秒のデータ転送が可能なSCIネットワークカード（**写真3**），異なるバスをメモリイメージで直結するバスアダプタ，PCIバスを拡張するエキスパンション，汎用インターフェースとしてファイバチャネル，ギガビットEthernetカードなどの展示が行われていた．

〔写真3〕ソリトンシステムズのSCIネットワークカード

〔写真4〕コンテックのフロントメンテナンスATXシャーシMPCシリーズ

PCハードウェアの診断ツールを展示していたのは，ウルトラエックスである．同製品では独自のセルフブートテクノロジを用い，OSを必要とせずにパソコンの電源投入後，数秒で起動し診断できる機能をもっている．必要となるシステムはFD1枚に収まっており，「P.H.D PCI」ではブート不可能なパソコンでも強制的にブートをかけ，診断する機能をもっている．

コンテックでは，フロントメンテナンスATXシャーシ「MPCシリーズ」の展示を行っていた（**写真4**）．同製品はスライドレールで主要部分を引き出して部品交換，ケーブル配線を全面に集中するなど完全フロントメンテナンスを実現している．また，オリジナルRASボードにより，WWWブラウザで各種設定や電源のオン/オフなどの遠隔操作が行えることも特徴である．

ケーメックスでは組み込みシステムやロボティクスに最適なCPUボードの新製品「ICP-P4」の展示を行っていた．同製品はCPUにPentium4を採用し，ハイパフォーマンスなシステム開発を強力にサポートするという．また，マルチディスプレイに高速，同時配信を可能にする高速データ転送システム「GigaSTAR」（**写真5**）も注目度の高い製品とのことであった．

〔写真5〕ケーメックスのGigaSTAR

昌新は，LANTRONIXの組み込み用シリアルデバイスサーバ「CoBox-Micro」の展示を行っていた（**写真6**）．サイズが40×49mmと小型であるため，ほとんどのシリアルデバイスに組み込むことが可能で，Ethernetへ接続するためのRJ-45コネクタを装備している．

〔写真6〕LANTRONIXのシリアルデバイスサーバCoBox-Micro

● 第3回 Bluetooth Expo

Bluetooth Expoは，Bluetoothの各種デバイス，測定機器，ソフトウェア，認証サービスからソリューションまでを幅広く展示しており，併設で「Bluetoothフォーラム」も開催された．

東陽テクニカでは，CATC社のBluetoothプロトコルアナライザおよびジェネレータが展示されていた．とくに新製品である「BT Tracer/Trainer」は，前機種から大幅に機能拡張されたフラグシップモデルであり，開発からテスト，評価までをカバーする統合開発環境を提供している．リアルタイムトレース機能，エラーパケット発生機能，HCIキャプチャモード機能が新たに追加されていることも大きな特徴である．

ネオテクノでは，Bluetooth実装に必要なハードとソフトをコンパクトにモジュール化した「BlueStick」が注目を集めていた（**写真7**）．ベースバンドLSI，プロトコルスタック部分を1チップCPUにソフトウェアとして組み込むことで，部品点数の削減，コストダウンを実現しているという．

〔写真7〕ネオテクノのBlueStick

ルネサステクノロジ（日立製作所半導体グループと三菱電機半導体が合併）では，Bluetoothベースバンド機能搭載マイコン「SH7630」，BluetoothベースバンドLSI「M64110WG」およびBluetooth RF LSI「M64846FP」の展示と，これらを利用したソリューションのデモを行っていた．

Show&News Digest

第2回UMLロボットコンテスト

■日時：2003年4月16日（水）
■場所：青山TEPIA（東京都港区）

オブジェクトテクノロジー研究所（旧社名：OMGジャパン）の主催により，UMLに関するカンファレンスUML Forum 2003が開催され，その中でUMLを用いた設計を競うロボットコンテストが開催された．競技内容は，以下の2種類．

- ショート・トラック競技部門

黒い線に沿ってトラックを一周し，その時間を競う．

- レスキュー部門

黒い線に沿って走ったあと，線のない区間を通り，グレースケール地帯，もう一度黒い線を走ったあと，城のオブジェのボタンを押すことによりお姫様と従者の人形を回収する"お姫様救出ゲーム"である．得点は時間および回収した人形の数により算出される．

完走率はショート部門で半数，レスキュー部門で3割程度であったが，ショート部門の優勝者は20mのコースを22秒代で走り，「20秒を切ることも夢ではない」という主催者側のコメントも聞かれる好結果となった．

また，会場の後方には各モデルのUML図が展示され，その設計を知ることができた．とくに審査員特別賞を受賞したチーム"AFlend"は，その状態モデルの着眼点が評価された．他のモデルは，マシンの状態を"直線モード or 曲線モード or 勾配モード"のようにorで表現していたが，AFlendだけが状態を"直線か否か and 勾配か否か"のようにandで表現しており，複数の条件が関わる現実事象をうまくモデリングした例として高く評価されていた．

UMLロボットコンテストのマスコット

審査員からは，各モデルに対して「昨年と比較して『とにかく正しく動かす』から『再利用を考慮して動かす』というレベルにまで進化している」という評価がなされ，コンテスト全体のレベルが向上していることが伺えた．

コースのようす

Richard Stallman氏 特別講演会

■日時：2003年4月21日（月）
■場所：虎ノ門パストラル（東京都港区）

Emblix主催により，GNUプロジェクトの推進リーダーであるRichard Stallman氏の講演会が開催された．

会場ではStallman氏によりGNUプロジェクト成立の背景と活動の紹介，GNU GPLの概念，Linuxはカーネルだけでなく GNUの成果物によりOSとして成立していることから"GNU/Linux"と呼んでほしいとのことのほか，近年のソフトウェアに対する特許戦略を挙げ，「すべてのプログラマに対し，特許問題は大きな脅威になりうる」と警鐘を鳴らした．また，「携帯電話などのソフトウェアもGPLにすべきか？」という質問に対し，「（ユーザーが）自由にソフトウェアを作成し，インストールすることができない組み込み機器は，GNUプロジェクトの範囲外ではないか」との意見が述べられた．

Richard Stallman氏

グレープシステム「ITRON入門と通信プロトコルセミナー」

■日時：2003年4月18日（金）
■場所：クイーンズタワーB（神奈川県横浜市）

（株）グレープシステムにより，ThreadX-μITRONとFusion通信プロトコルスタック，Webプロダクトに関するセミナーが開催された．

ThreadX-μITRONは組み込み向けリアルタイムOSであるThreadX上にμITRON互換レイヤを搭載したもので，これを題材にμITRONのタスク管理機能，スケジューリング，同期通信，割り込み，OSコンフィギュレーションなど，μITRONのプログラミングに関して留意すべき一般事項がわかりやすく解説された．

続く通信プロトコルスタックであるFusion NETは，TCP/IP，PPP，SNMPなどのプロトコルスタックについて，その概要と移植の方法などを実際のコードを交えて解説した．

また，組み込み向けXMLツールキットFusion WEBについても取り上げられた．なかでもFusion XMLマイクロパーサはANSI Cで書かれたXMLパーサ．通常，この種類のソフトはJavaやC++などで書かれているが，組み込み向けということもありANSI Cを採用し，ROM 13Kバイト，RAM 2Kバイトで動作する小サイズが特徴とのことだ．ほかにもXMLスキーマコンパイラ，SOAPプロトコルスタックなども紹介された．

セミナーのようす

Jaluna OSセミナー

■日時：2003年4月11日（金）
■場所：JAFCOホール（東京都千代田区）

昨年10月にJaluna社と業務提携したウェブソフト・インターナショナル（株）により，Jaluna OSに関するセミナーが開催された．

Jaluna OSはC5マイクロカーネル（Chorus OS Ver.5）をベースとした分散リアルタイムOS．RT-POSIX APIをサポートし，カーネル本体のサイズは30Kバイト，システムを構築するうえでのトータルでの最小フットプリントは60Kバイト～．メモリマネージャやスケジューラなどはカーネルの外部に位置する．サポートCPUはPowerPC/x86/SPARCで，MIPSやARMに関しては準備中とのことだ．

JalunaOSは任意のプロセスをユーザーモードおよびカーネルモードで動作させることが可能という特徴をもつ．そのため，製品開発時にはユーザーモードでデバッグを行い，出荷時にはカーネルモードで動作させ速度を向上させることなどが可能だ．

JalunaOSはマイクロカーネルベースのOSをそのまま利用するJaluna-1と，その上にLinuxエミュレーション環境を導入してLinuxのアプリケーションとリアルタイムOSの性能を同時に使用可能とするJaluna-2の二つの利用形態が選択できるようになっている．また，VxWorks Compatibility Kitも提供される．

Jaluna-1 Developers Editionはオープンソース，ロイヤリティフリーで提供され，SourceForgeでも公開されている（http://jaluna.sourceforge.net/）．

ハッカーの常識的見聞録

31

広畑由紀夫

今月の常識

Windows Serever 2003 製品版がやってきた！

☆4月25日．ついに米国でWindows Server 2003が発売されました．さらに米国ではVisual Studio .NET 2003も発売開始となり，いよいよサーバ製品から開発にいたるまで，.NET製品が出そろったことになります．今回は，その進化のぐあいを見てみたいと思います．

● Windows Server 2003

2003年4月25日に発売開始されたバージョンは，米国内向けの英語版ですが，すでに日本語版も製造工程に入っており，正式な発売日程は発表されてはいないものの，早ければ本号が発売されるころには発売開始になっているかもしれません．さて，本当に長く待たされたWindows Server 2003ですが，製品版で筆者がとくに気に入った点を紹介します．

1）サーバモードのCPUリソース割り当てがさらに重視されている

既定のインストールが終了した時点で，サーバOSなのですから当然のごとくサーバ用に動作が設定された状態になっています．この状態では，Direct3D拡張機能はおろか，クライアントの動作速度に影響するほどのCPUリソースをバックグラウンド処理に割り当てているようです．

2）POP3サービスが含まれるようになった

従来からSMTPサービスは含まれていましたが，Windows Server 2003スタンダード版においてもPOP3サービスが追加できるようになったようです．いままでWindowsで標準のメールサーバといえばExchange Serverで，別途に購入を強いられるという印象が強かったのですが，SMTP/POP3サービスが追加インストール可能(既定ではインストールされない)になったことで，標準のみでメールサービスからWebサービス，さらに接続規模を重視しなければMSDEなどのデータ接続なども利用できるようになりました．

接続規模やパフォーマンス性能を重視する場合は，メールサーバであればExchange，データベースであればSQL Serverなどのマイクロソフト製品を選ぶか，他社製品を選ぶかを選択できるわけです．少なくともいままでのように，「メールサーバを立てたいけれど標準でPOP3がないからLinux……」という簡単な話では済まなくなりそうです．

3）アプリケーション(クライアント)モードが速い

筆者の環境ではバックグラウンドモードからシステムの設定をアプリケーションモードに切り替えると，バックグラウンドモードの速度が嘘のように軽くなりました．さらに，評価版で使用していたときと比較しても，同一環境にもかかわらず速くなっていると感じられるほどの最適化がなされているようです．

この感覚を実際の数値として示してみたかったのですが，4月末現在，公式な情報として公開されていないので，導入する機会がある場合には，ぜひとも同一の環境で以前のバージョンとの動作速度を比較してみてください．物理メモリが十分にある場合には，より速くなったと実感できる人は多いだろうと思います．

4)既定の設定で非常にセキュリティ重視となっている

既定のインストールでは，サーバ用拡張サービスは「ファイル共有サービス」のみになっています．その他のサービスが必要な場合には，「サーバの役割管理」でサービスを追加していくように設定されています．もちろん不要なサービスはインストールせずに済むので，初期インストール後にサービスを停止して回るようなこともなくなりました．

クライアントレベルでも，WebのフロントエンドとなるInternet Explorerの初期設定では，クライアント用として設定されていないため，ActiveXコンポーネントはおろか，ほとんどのスクリプト処理が禁止もしくは使用不可となっており，ダウンロードさえもサイトを登録してから行うようになっています．こうした制限のため，Windows Updateすらも，使用するためには設定を行わなければならないようです．

Windows Updateに関しては，個々のサーバで行うよりも，Windows Updateを中継するSoftware Update Serviceを使用したほうがよいのかもしれません．

● 今後の展望

Visual Studio .NET 2003は，4月末現在，日本語版の製品出荷は未定ながら，すでにMSDN会員向けにダウンロード配布が開始されています．開発現場に対してはすでに製品の導入が始まっています．筆者も開発環境の移行を済ませました．Visual Studio .NET 2003については，稿を改めて紹介する機会もあるでしょう．

マイクロソフトの公式Webページでは，Officeとの連携に関わる部分の開発を統合するとのロードマップが公開されています．Office System 2003は4月末現在で第2ベータの段階であり，今後の開発で変更される部分もあることでしょう．Windows Server 2003との連携で発表されているShare Point Portal Serviceやグループウェアなどは，Office System 2003で統合されて発売されるとのことなので，こちらも今後に期待したいと思います．

ひろはた・ゆきお　OpenLab.

ロゼッタストーンとWWW

山本　強

いつの頃からかは知らないが，最近のブラウザは基本的に多言語対応となっており，見るだけならほとんどの外国語を扱うことができる．以前はWWWをさまよっていると文字化けしたWebページに遭遇してびっくりすることがよくあったが，最近では中国語やハングルであっても文字まで正しく表示されるのが普通になっている．それはそれで素晴らしい技術革新だと思うのだが，実際のところ日本語と英語ぐらいしか理解できない平均的な利用者には，そのありがたさが伝わってこない．

まぁ，そんなものかと思っていたある日，Yahoo!のトップページを眺めていて，その末尾に各国版Yahoo!へのリンクがあることに気が付いた．試しに，中国語版Yahoo!をクリックして驚いた．なんと日本語版Yahoo!と同じようなデザインの中国語版Yahoo!のトップページが表示されるのだ．誤解を避けるためにいうと，筆者は多言語対応に驚いたのではなく，日本語版と中国語版とのデザインの類似性に驚いたのである．筆者は，とっさにロゼッタストーンを連想してしまった．

アイコンで理解する基本単語

中国語では，外来語も基本的に漢字で表記する．コンピュータなら電脳，携帯電話なら手机あるいは手機と書く．こういった基本単語の場合はだいたいわかるのだが，もう少し漠然とした概念語となるとわからない．たとえば，モバイルである．この場合，モバイルというものがあるわけではなくて，移動体通信を使った情報サービス全体を意味している．これをほかの言語ではどう表記しているのだろうか．

現在の各国版のYahoo!のトップページは，最上段に何個かアイコンが並んでいるデザインになっている．アイコンにはそれが何を意味するかの文字が付記されているから，同じデザインのアイコンに付記されている単語は基本的に同じであると考えられる．日本語版Yahoo!の携帯電話アイコンにはモバイルと注釈されているから，まず中国語系Yahoo!を調べると同じアイコンが二つ見つかった．そして，そこには漢字で以下の添え字がしてあった．

短信（中国簡体字版，cn.yahoo.com）

簡訊（台湾版，tw.yahoo.com）

なんと，大陸と台湾では言い方が違っているらしい．短信は，「手短なメール」といった意味合いなのだろう．簡訊は，「簡単に情報入手」といった雰囲気が感じられる．勢いで欧米語圏を調べてみたところ，フランス，イギリスをはじめとして多くはMobileであった．

変わったところでは，イタリア語はCellulari，ドイツ語はHandyと表記してあり，多少の自己主張をしているようである．

このように，トップページから読み取れる中国語インターネット用語にはなかなかの味わい深いものがある．チャット＝聯天室，オークション＝拍売，株式市場＝股市などがアイコンから読み取れる．株＝股というのは日本語的にはびっくりもので，米国株式新聞は中国語では美股新聞となり，なんとも怪しくなってしまう．

アイコンはリンク先の内容を抽象化したものだから，その先のページに出てくる用語の意味もだいたいわかってくる．先の「短信」の先を見ると，もっとユニークな業界用語が出てきた．携帯電話の着メロは「手機鈴声」と表記するらしい．まだまだあるのだが，この先は読者の練習問題として残しておこう．

サーチエンジンも多言語対応

Yahoo!のようなリスト型の情報サービスで，単語のだいたいの意味合いが読み取れるのだが，自分が理解している意味が正しいかどうかを確かめる方法が必要である．そこで出てくるのがGoogleなどの検索エンジンである．実際に試してみて驚いたのだが，Googleは中国語と日本語の壁を越えて検索してくれる．検索エンジン内では中国語と日本語の文字の区別がなされていないらしい．Unicodeはそういうコード体系だから，Unicodeで内部データベースを作っていれば自然にそうなるのである．

Yahoo!のアイコンや分類でだいたいの意味がわかったら，今度はそれをGoogleで探索して用例を確認できてしまう．もちろん，確認するためには中国語の知識がある程度必要になるが，まれには英語や日本語の対訳が付いているのである．Unicodeによる多言語文字の統一は，ローマ字文化圏による言語文化侵略のようにいわれることが多いのだが，このように言語をまたぐ検索ができるようになるというご利益もあるのである．

＊　　　＊

今回の話題は，Yahoo!も見方によってはロゼッタストーンに見えるというところから始まったのだが，なぜこんな使い方ができたのだろうか．何のことはない，アイコンという画像に必ず注釈がテキストでついていたということなのである．Yahoo!のデザインはテキスト中心になっている．すべての画像に解説文字をつけるというのは，Webページのユニバーサルデザインでは基本中の基本である．だから，多言語の関連付けができたともいえる．やっぱり基本は大事である．

やまもと・つよし　北海道大学大学院工学研究科電子情報工学専攻
計算機情報通信工学講座　超集積計算システム工学分野

特集 高速ロジックインターフェースから PCI Express, 10Gigabit Ethernet まで

高速バスシステムの徹底研究

USB や Ethernet のようなインターフェースから PC/AT 互換機のメモリバスや CPU の FSB など，あらゆるバスインターフェースは高速化の一途をたどっている．これらは，デュアルエッジでデータを転送したり，低電圧化などの電気的な改良などの積み重ねにより実現されている．また，バスをシリアル化，または狭バス幅化を採用するシステムも増えてきている．

さまざまなバスインターフェースはどうやって高速化することができたのか，それを理解するにはまず，もっとも基本的なデバイス間通信の高速化技術を理解する必要がある．その技術の延長線上に，バスインターフェースの高速化が実現できていることがわかるであろう．

本特集では高速バスシステムというキーワードから，デバイス間通信，ボード間通信，筐体間通信，システム間通信など，用途や距離に応じたそれぞれの分野でよく使われている，またはこれから普及すると思われるバスインターフェースを取り上げて解説する．

Prologue

高速バスいろいろ
―― デバイス間データ転送からボード間/筐体間通信まで

桑野雅彦

● PCのFSB，最近やたらと速くなっていませんか？

高速化の変遷がもっともよくわかるものの一つに，PC/AT互換機があります．CPUが年々速くなっている点が目に付きますが，ここでは外部バス，いわゆるFSB（Front Side Bus）に注目してみましょう．

クラシックPentiumでは66MHzだったFSBが，Pentium IIIでは133MHzになり，最新のPentium4では800MHzというものまで登場しています．ただしPentium4の場合には，800MHzといっても1クロックあたり4回のデータ転送が行われるので，実際のクロック信号の周波数は200MHzですが，それでも高速化されていることには変わりありません．

しかしマザーボードに使われている基板は，FR4と呼ばれる昔から使われている一般的なもので，あれだけ多ピンのQFPやBGAを使っているのに，4層基板が普通なのです．CPUのクロック周波数ほどではないにしても，材質的に昔とそれほど変わらないプリント基板で，なぜこれほどの高速化が実現できたのでしょうか．その謎を解き明かすのが，今回の特集です．

● デバイス間データ転送の高速化

今回の特集の第1章では，ロジック回路の高速化について解説します．さきほどPC/AT互換機の例を説明しましたが，たとえばメモリモジュールを考えてみましょう．昔のSIMMは5V電源のDRAMでした．それが3.3VのSDRAM DIMMになり，DDR-SDRAMでは電源電圧が2.5Vになり，さらに信号振幅も低電圧化されています．

そしてさらなる高速化手法として，Gbpsオーダのデータ転送では，差動伝送方式が使われています．

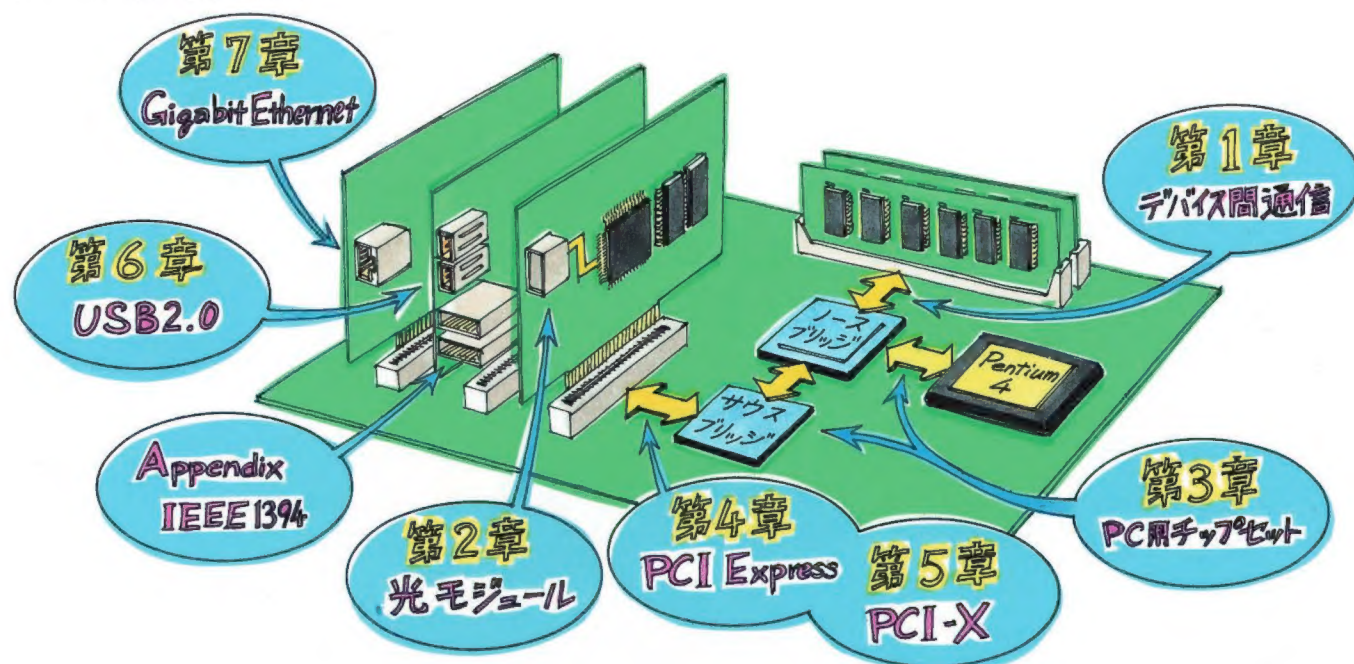
メモリデバイスとチップセットなど，デバイス間データ転送の高速化のキーワードは次のようなものになるでしょう．

- 電源電圧の低電圧化
- 信号振幅の低電圧化
- 差動伝送化

● 半導体製造技術の進歩が高速化を支える

第1章で詳しく解説していますが，高速化のキーワードとしてあげた差動伝送などは，じつは昔から使われていた技術なのです．ただし，昔は通常ロジック回路と差動伝送回路は別々のブロックに分かれ，その間には変換回路を必要としました．

それが現在では，半導体製造技術の進歩により，一つのデバイスの中に実装することが可能になりました．これにより，高

〔イラスト〕特集で解説する分野

Column
特集で取り上げた以外の高速インターフェース

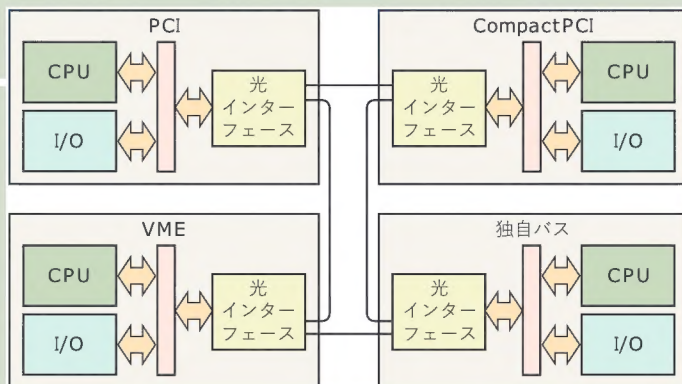
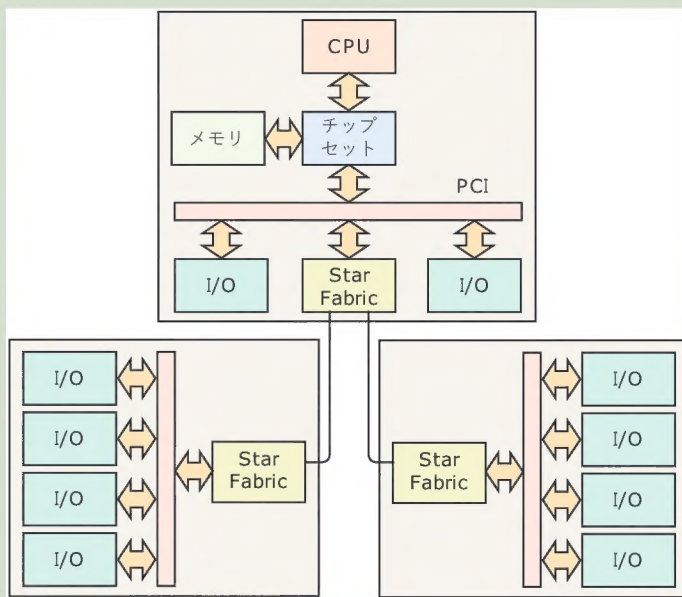
高速拡張バスやインターフェースには，今回特集で取り上げたもの以外にもさまざまなものがあります．

ストレージ系としては，従来のIDEをシリアル化したシリアルATAが普及し始めています．またサーバ用途では，ファイバチャネルと呼ばれるストレージインターフェースも使われています．

またFA用途では，規格化された標準バスや標準インターフェースにしばられることなく，独自仕様のインターフェースで筐体間通信を行っているものなど多数存在します．

たとえば，**図A**のようにPCIやCompactPCI，VMEなどのシステム間を，光ファイバで接続するPCIやVMEボードが市販されています．内部的にLANなどのシリアル通信デバイス形態を採るものや，共有メモリ空間をもちデータ転送はすべてハードウェア的に行われ，ソフトウェアはメモリのリード/ライトを行えばすむものなど，さまざまなスタイルの通信インターフェースボードが市販されています．

また，FA用途では多くの拡張スロットを必要とする場合もあります．そのような用途にはPCIと互換性のあるStarFabricなども普及し始めています．StarFabricのブリッジはソフトウェア的にはPCI-PCIブリッジに見えるので，**図B**のようにホストシステムと拡張スロットボックス間をStarFabricで接続した場合でも，従来のソフトウェアをそのまま使うことができます．これにより，従来のPCI-PCIブリッジでは実現できないような数のPCIスロットを用意することもできます．またStarFabricはLANのハブのようなスイッチデバイスもあるため，非常に柔軟な接続形態を採ることができます．

〔図A〕光接続による筐体間通信

〔図B〕StarFabricによるPCIバスの拡張

速データ転送に対応できるデバイスを安価に製造でき，1チップ化により基板面積も縮小できるなど，さまざまなメリットを生み出しています．

USB2.0やPCI Expressなどには差動伝送技術が使われていますが，これは通常ロジック回路と差動伝送回路を，1チップで安価に作れるようになった今だからこそ実現できるバスインターフェースではないでしょうか．

● バス/インターフェースの高速化

ここでデータバスの高速化を考えてみましょう．単純に考えれば，バス幅を広げ，転送クロック周波数を上げれば，それだけで高速化を実現できます．しかしこの方法では，基板配線遅延などの問題で，クロックに対してデータの到着がばらついてしまい，高速化が難しくなるのです．これをふまえ，現在ではバス幅を狭くしたりシリアル化を採用するバスシステムが増えてきています．

さらにクロックスキューの問題を解決するため，データとクロックを別々に送るのではなく，データにクロックを埋め込んで1本の線で伝送する方法も採用されています．

バス/インターフェースの高速化のキーワードは，次のようなものになるでしょう．

- 狭バス幅化
- シリアル化
- クロック埋め込み

● 特集の案内

今回の特集で解説する内容はイラストのとおりです．

一見すると，ボード上のデバイス間通信と，筐体間をケーブルで接続するインターフェースは，まったく別物に見えます．しかしそこで使われている高速化技術には共通するものがあり，いわゆる陸続きであることを忘れてはなりません．

くわの・まさひこ　パステルマジック

第1章 LVTTL/SSTL/HSTLなどのシングルエンドから，LVDSなどのディファレンシャルまで

高速ロジック回路の電気的仕様いろいろ

井倉将実

高速バス/高速インターフェースを正しく理解するには，電気信号を高速に送るための基本的な知識が必要になる．本章では，第2章以降で取り上げる各種バスやインターフェースを理解する上で必要となる，高速ロジック回路の電気的特性などについて解説する．

（編集部）

信号インターフェースのいろいろ

● さまざまな信号インターフェース

現在，身の回りにある半導体デバイスのインターフェースには，どのようなものがあるのでしょうか．

5V電源のTTLやCMOS，3.3V電源のLVTTLやLVCMOS，さらにはDDR-SDRAMなど電源電圧の低いデバイスも多く使用されています．**表1**は，筆者が本誌2003年1月号で紹介したSH-4搭載CPUボードの場合ですが，5VのTTLや3.3VのLVTTL/LVCMOS，そしてCPUやFPGAのコア電圧はさらに低いものになっています．

ほんの数年前は，TTLインターフェースとCMOSインターフェースの2種類さえ知っていれば，ほとんどのロジック回路の設計は事足りていましたが，それではせいぜい100MHz程度のバスクロックが限界でした．これを超えようとすると，TTLやCMOSインターフェースでは対応できず，別の信号インターフェースが必要になります．

さらに，ここ1～2年ほどの間に，一気にGbps転送帯域までの要求が増えてきたように思います．このクラスになると，LVDS（Low Voltage Differential Signaling）に代表される差動インターフェースが必須です．筆者自身，いろいろなシステムの設計において，どのようなインターフェースを採用して信号伝送を行うか，いろいろ方式を検討する場面が増えてきました．

例として，標準的なマイコン（外部バスはLVTTL）に何らかの高速なメモリインターフェースを採用した製品の設計を想定してみましょう．一般的なマイコンはLVTTLをサポートしているので，そのままメモリとしてSDRAMを接続することは可能です〔**図1**（**a**）〕．

しかし，SDRAMよりもさらに高速なメモリであるDDR-SDRAMを採用する場合には，DDR-SDRAMのインターフェースであるSSTL2（SSTL：Stab Series Terminated Logic）のサポートが必須です．これはLVTTLとは明らかに異なる信号であるため，LVTTLのシステムにそのまま接続することがで

〔**表1**〕
筆者の設計したSH-4システムの信号系

デバイス	電　圧	規　格
SH-4（コア）	1.95V	−
SH-4（I/O）	3.3V	LVTTL3.3
FPGA（コア）	2.5V	−
FPGA（I/O）	3.3V	LVTTL3.3
SDRAM	3.3V	LVTTL3.3
PCI	3.3V	3.3V PCI
PIO	3.3V	LVCMOS3.3
RTC	5.0V	TTL

〔**図1**〕**異なるインターフェースをもつデバイスを接続するには**

（**a**）LVTTLのみ　（**b**）DDR-SDRAMをつなぐ場合

きません〔**図1**(**b**)〕.

● 進化するインターフェース

本章で紹介する信号インターフェースは，技術の進歩とともに種類が増えてきたものです．たとえば，差動インターフェースの代表であるECLは，低電圧化の時代の波にのってPECLに代わり，さらに低電圧化してLVPECLに進化しました．おなじみのCMOSインターフェースも，現在ではLVCMOS1.8のように低電圧化されています．

今後，技術の進歩により，さらなる信号インターフェースが提唱されることになるでしょう．そこで，次に現在の一般的な信号インターフェースのおさらいをしておくことにしましょう．

● シングルエンド/ディファレンシャル

広く利用されているTTLインターフェースやCMOSインターフェースは，グラウンドを除けば1本の線で信号を伝送する1線式です．

しかし，Ethernetの信号伝達に使われるCML(Current Mode Logic)や，超高速なクロック供給回路などで広く使われているPECL/LVPECL，そしてLVDSなどは2線式です．

とはいえ，設計現場で「1線式インターフェース」や「2線式インターフェース」などと呼ぶことはありません．

TTLやCMOSのように，1本の信号線で伝送する方式を「シングルエンドインターフェース」，対してLVDSなどのように2本の信号線で伝送する方式を「ディファレンシャルインターフェース」と呼びます．

信号インターフェースは，大きくこの2種類の伝送線路方式に分類でき，取り扱い方法や使用帯域などが大きく異なります．また，シングルエンドとディファレンシャルでは，信号の伝わり方の考え方がまったく異なります．

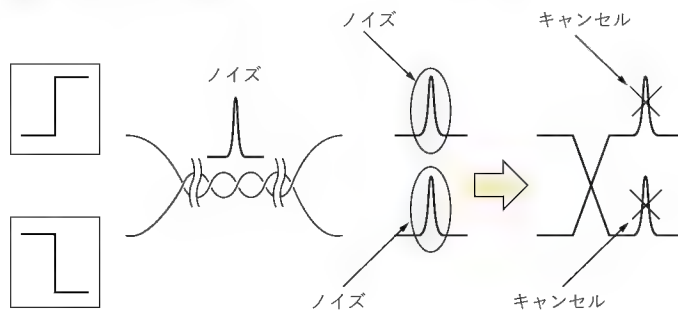
● シングルエンドの信号の伝わり方

ロジック回路では，伝送線路が1本で伝わるシングルエンドの場合には，信号線以外に必ずグラウンドが必要です．そして，信号は「ある電圧から何ボルト以上あれば"H"レベル」，「何ボルト以下であれば"L"レベル」となります．

この「ある電圧」をスレッショルドレベルと呼び，TTLやCMOSの場合の電圧基準としてグラウンドレベルを使います．この場合のグラウンドレベルは，別に0Vでなくてもかまいません．

信号を発生させるドライバ側と信号を受け取るレシーバ側の共通電位の電位差が0Vであれば，たとえほかの電源からみて100Vのオフセット(＝上乗せ)があっても，それはグラウンドレベルになります．

このように，グラウンドレベルと信号線との電位差によって，信号が"H"レベルなのか"L"レベルなのかを伝える方式が，シングルエンドインターフェースの方式です．

ただし，高速なシングルエンドインターフェースになるとまた事情が変わります．DDR-SDRAMやRAMBUS-DRAMのインターフェースであるSSTLやHSTL(High Speed Transceiver Logic)では，信号線とグラウンドレベルとの電位差だけでなく，基準電圧(V_{ref})と比較して何ボルト以上の電位差があるかによって"H"レベルと"L"レベルを決めます．詳細は後述します．

〔**図2**〕外乱ノイズに強い

● ディファレンシャルの信号の伝わり方

ディファレンシャルインターフェースは，一つの信号伝送のために，必ず2本の信号線が存在します．

ディファレンシャルインターフェースの「ディファレンシャル」とは「差(Differential)」を表していますが，何の差かというと，2本の信号線間に生じる電位差を示しています．そして，この2信号線間の電位差をみて，"H"レベルか"L"レベルかを判定します．

ディファレンシャルインターフェースは，必ず一つの信号につき2本の信号線が必要になるということで，配線数が増えてしまいます．たとえば，8ビットバス幅の信号伝送をするためには，シングルインターフェースの代表であるLVTTLインターフェースでは，GND信号と実際の信号線8本の合計9本で済みますが，ディファレンシャルインターフェースでは8ビット×2対で16本の信号線が必要です．また，グラウンド信号は必ずしも必須ではありませんが，通常は基板間/筐体間の基準電位を確保するために必要です．

● ディファレンシャルのメリット

シングルエンドインターフェースと比較すると，信号線本数が多くなるというデメリットのあるディファレンシャルインターフェースですが，そのデメリットに対してはるかに多くのメリットが存在します．

まず，各種ディファレンシャルインターフェースに共通する大きな特徴の一つが，外乱ノイズに強いという利点です．

図2は，外部からあるパルスノイズが，より線(ツイストペア)のケーブルに乗ったことを想定しています．パルスノイズは，両方の信号線に同一時刻に同電位分が上乗せされますが，受信端はまったく影響ありません．

これは，両端子の電位差によりレベルを判定するというレシーバの基本的な動きを考えればよいのです．パルスノイズ電圧の電圧をV_{noise}とすると，差動信号線の片側の信号線P端子上の電圧は，$V_a + V_{noise}$です．同様に，もう片側のN端子上の電圧は，$-V_a + V_{noise}$になります．ここで，差動レシーバの動き

を適用します．出力信号レベルは，$V_P - V$ なので，

$$(+V_P + V_{noise}) - (-V_N + V_{noise})$$
$$= +V_P - (-V_N) + V_{noise} - V_{noise} = V_P + V_N \quad //QED$$

という結果になり，V_{noise} の影響は消えます．これを「同相電圧除去」と呼びます．どの程度この同相電圧を取り除くことができるのかという性能を，アナログ OP アンプなどのスペックでは「同相電圧除去比」といい，〔CMRR：Common Mode(noise) Reduction Ratio〕と記されている場合もあり，dB（デシベル）で表記されています．

上記の式は，CMRR が無限大の 100%理想的な差動レシーバの動きを考えたものであるため，現実的にはここまで完璧に V_{noise} の影響をキャンセルできません．しかし，簡単な差動レシーバでも，CMRR は 20dB = 100 程度の性能はあるので，V_{noise} の影響を 100 分の 1 にすることができます．

外乱ノイズは，何も外部からやってくるだけとは限りません．グラウンドプレーン上にディファレンシャルインターフェースを使ってデバイス間を接続したとき，このグラウンドレベルが他のデバイスの同時スイッチングの影響により数 100mV 程度のノイズが瞬間的に走ることもあります．このようなときに，同相電圧除去特性により，純粋な信号成分だけを取り出すことができるのも，ディファレンシャルインターフェースの大きな特徴でしょう．

Column 1

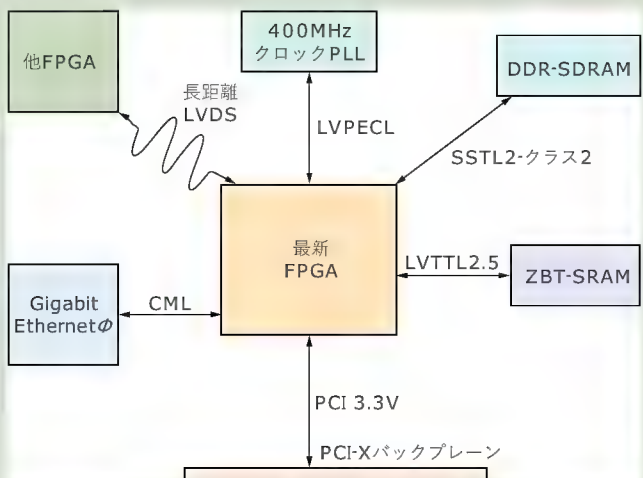
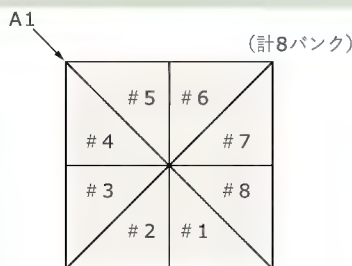
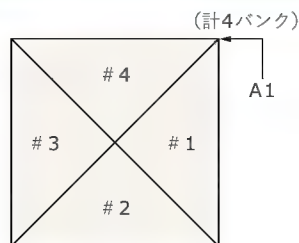
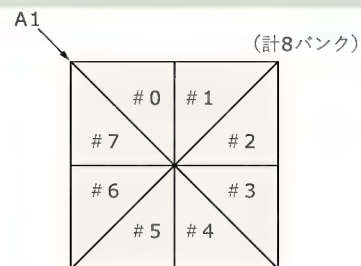
FPGA が広める高速インターフェース?!

数年前まで，FPGA を採用して数百 Mbps のデータ転送を行いたい場合，かりに ECL/PECL インターフェースを採用しようとしても，普通に入手できる FPGA には直接接続できず，FPGA 以外に MECL100KH など（TTL でいう 74xx ファミリと同様なもの）の ECL-SSI/MSI デバイスをいくつも並べて論理回路を構成するしかありませんでした．または，TTL ⟷ PECL 変換インターフェースデバイスがあったので，これを採用するという方法もありましたが，それでも基板面積の増大は避けられませんでした．

しかし，最近の高性能 FPGA は「ここまでヤルか！」というぐらいに多彩なインターフェースをサポートしています．これを実現するのに，このエリアは LVTTL，このエリアは LVDS というように，FPGA の I/O ピンに「バンク」という概念をもたせ，各バンクごとにインターフェースを切り替えて対応しているのです（**図 A**）．

これにより，Gigabit Ethernet の物理層と DDR-SDRAM メモリ，そして PCI バスインターフェースを一つの FPGA に直結するというようなこともできます（**図 B**）．実際に，このような使い方は筆者の場合は日常茶飯事であり，当初はバンクという概念が面倒に思えていましたが，いろいろな信号インターフェースに対応する手法としてはよく考えられているなと感心しています．

このように，少し前までの FPGA を使ったロジック設計では見向きもしなかった（？）信号名称が，最近の高性能 FPGA では自在に使えるようになり，それまでそれらのインターフェースを使ってこなかった設計者や新人設計者が慌てているという話も聞きます．今回解説している ECL や CML などは 20 年以上も前から使用されており，筆者にとっては新しいインターフェースでも何でもないと感じているのですが……．

〔図 B〕さまざまなインターフェースを一つの FPGA に接続

〔図 A〕最新 FPGA のバンク構成

シングルエンドインターフェース編

はじめに

10年ほど前までは5V駆動のICが一般的であり，ようやく3.3V系のドライバ群が出始めたところなので，デバイス間の接続で注意することは電位差だけでした．

しかし，ここ数年で低電圧駆動のデバイスの開発が進み，新しい信号規格がいくつも出てきています．

ここでは，比較的よく使われているシングルエンドインターフェースについて解説します．

LVTTL

● LVTTLとは

LVTTLは，CMOSインターフェースと並んで，ICの接続インターフェースのもっとも基本であるTTLの低電圧版です．

TTLインターフェースは，5V系のIC接続では非常にポピュラーな信号インターフェースです．しかしながら，近年になり，ICの駆動電圧が低電圧になってきたのにあわせて，信号インターフェースも低電圧化の要求がどんどん高まりました．そこで，旧来の5V系TTLインターフェースとある程度互換が取れるような信号インターフェースの規格化が必要となり，TTLインターフェースの低電圧化がはかられました．それが，このLVTTLインターフェースです．

LVTTLというと，通常は3.3V系のTTLインターフェースを指します．「通常」というのは，この3.3V LVTTLが登場してからすでに10年近くが経ち，3.3V LVTTLでさえも時代遅れになりつつあるからです．

現在では，2.5V系のTTLインターフェース，さらには1.8V系のTTLインターフェースが登場し始めました．

各社で呼称が違うということもありますが，複数の低電圧駆動系に対応したLVTTLインターフェースということで，LVTTLという名称の後に，駆動電圧を入れる場合があります．現在のところ，「LVTTL3.3」や「LVTTL2.5」，また「LVTTL1.8」の3種類を目にします．

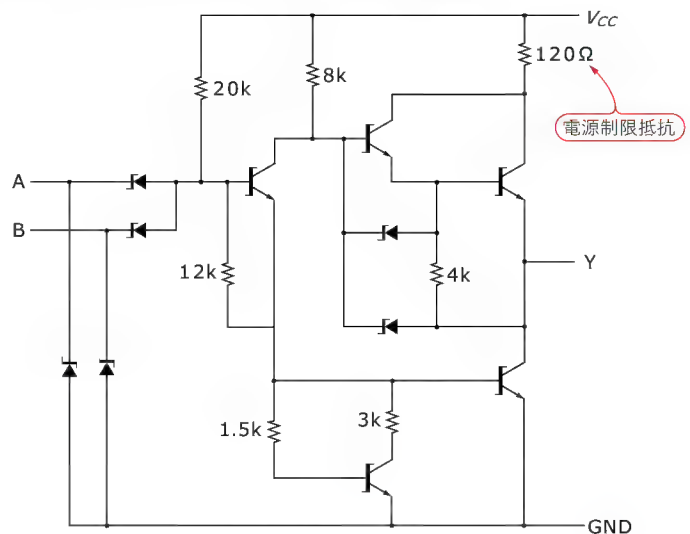
● LVTTLの特性

基本となるTTLインターフェースの入出力部分の回路を**図3**に，またTTLの特性表を**表2**に示します．この表は，あるFPGAのI/Oピンを，LVTTLインターフェースにした場合のいくつかの重要な特性パラメータを表しています．

- V_{OH} ：“H”レベル出力時の最低出力電圧
- V_{OL} ：“L”レベル出力時の最大出力電圧
- V_{IH} ：“H”レベルとして判定する場合の入力最低電圧
- V_{IL} ：“L”レベルとして判定する場合の入力最大電圧

新人技術者の養成研修などの際に，特性パラメータ表に最大値や最小値が明記されていないことについて疑問をもたれ，質問を受けることがあります．

たとえば，3.3V LVTTLのV_{OH}パラメータの場合，最低出力電圧である2.0Vはわかるが，最大時(MAX)が明記されていないという点です．この3.3V LVTTLの場合には，V_{OH}(min)で

〔図3〕TTLインターフェースの入出力部分の回路

〔表2〕TTLインターフェースの特性（LVTTLの特性はアルテラ社Stratixのデバイスデータシートより）

パラメータ	74LS00			LVTTL3.3			LVTTL2.5			単位
	最小	標準	最大	最小	標準	最大	最小	標準	最大	
V_{CC} ：動作電圧	4.75	5.00	5.25	3.00	3.00	3.60	2.38	2.50	2.63	V
V_{IH} ：Hレベル入力電圧	2.00			1.70		4.10	1.70		4.10	V
V_{IL} ：Lレベル入力電圧			0.80	−0.50		0.70	−0.50		0.70	V
II_{H} ：Hレベル入力時リーク電流			0.02	−0.01		0.01	−0.01		0.01	mA
V_{OH} ：Hレベル出力電圧	2.70	3.40		2.00			1.70		2.10	V
V_{OL} ：Lレベル出力電圧		0.25	0.40			0.45	0.20		0.40	V
I_{OH} ：Hレベル出力時電流			−0.40			−25.00			−25.00	mA
I_{OL} ：Lレベル出力時電流			8.00			25.00			25.00	mA

規定された以上の電圧を出していれば，それで動作が保証されるからであり，厳密に最大値を知ろうとした場合には，信号インターフェースを出力するIC内部の出力I/Oパッドの構造が公開されていないかぎりわかりません．

ですが，どんなに最大の電圧を出そうとしても，I/Oピンに印加された駆動電圧までしか電圧を持ち上げることはできないので，そういう観点でいえば「最大値は3.3Vです」といっても間違いではありません．

しかし，すべてのICが最大3.3Vを出力できるかといえば，これも正しくありません．

先の**図3**で示したように，出力段がトーテムポール構造になっている場合には，トランジスタのV_{CE}電圧分があり，そして普通は電流制限抵抗(図では120Ω抵抗)があるので，過電流が流れた場合にはこの抵抗×電流値分＋トランジスタのV_{CE}分の電圧ドロップが必ずあります．結果的に，出力パッドに現れる電圧は3.3Vよりも低くなります．

● TTLデバイスとLVTTLデバイスの接続

また，旧来から使われ続けている5V系TTLデバイスと3.3V系TTLデバイスは，V_{IH}に要求される電圧がほとんど同じなのですが，それではマージンが小さいために，双方を接続した場合には信号伝送を100%保証することができない場面も少なくありません．

ここで再度，**表2**のSN74LS00についての電気的特性を見てください．SN74LS00は，5.0V電源で駆動します．V_{OH}/V_{OL}/V_{IH}/V_{IL}の各4パラメータは，いずれもLVTTL 3.3V系に合致します．ですが，余裕度(マージン)を考えると，あと一息ほしいところです．

たとえば，5.0V系TTLがV_{IH}=2.0V(MIN)に対して，3.3V系LVTTLがV_{OH}=2.4V(MIN)ぐらいあれば，400mV程度の余裕度，つまり20%のマージンがあるので，よほどのことがない限りおかしくなることはないでしょう．

そこで，必要に迫られて昔の回路をそのまま使わなければならないような場合は，次に紹介する3.3VのLVCMOSや2.5VのLVCMOSを使うことをお勧めします．

〔表3〕CMOSインターフェースの特性

パラメータ	74VHC245(Ta＝25)			単　位
	最　小	標　準	最　大	
V_{CC}：動作電圧	2.00		5.50	V
V_{IH}：Hレベル入力電圧	1.50			V
V_{IL}：Lレベル入力電圧			0.50	V
II_H：Hレベル入力時リーク電流			0.10	μA
V_{OH}：Hレベル出力電圧	V_{CC}× 90%		V_{CC}	V
V_{OL}：Lレベル出力電圧	0.00		0.10	V
I_{OH}：Hレベル出力時電流			－25.00	mA
I_{OL}：Lレベル出力時電流			25.00	mA

LVCMOS

● CMOSとは

CMOSの最大の特徴は，出力信号電圧がI/Oパッド駆動用の電源電圧の全範囲に振幅する点です．また，入力の電位も，電源電圧のちょうど中心電圧で切り替わります．

入力ピンの内部構造は，一般的にPチャネルMOS-FETとNチャネルMOS-FETの組み合わせであるCMOS構造です．出力ピンも同様にMOS-FET構成である場合が一般的です．

電気が流れることによる電界効果によって素子のON/OFFを行うFETは，電流が流れることによって素子のON/OFFを行うトランジスタと比べて，理論上は電流が流れないという利点があります．

電流が流れないということは非常に抵抗値が高いということであり，そのため消費電力はほぼゼロです．通な言い方(?)をすれば「入力インピーダンスが非常に高い」となります．

それゆえ，LVTTL信号インターフェースに比べて，入力ピンに何も接続されていない状態(フローティング)では，外部からのノイズなどで容易に誤動作を起こすという症状も現れます．

同様に，高入力インピーダンスということは，入力側に接続されるデバイスが小電流しか発生できない場合にも，ロジックとして接続できるという利点もあります．よって，フォトセンサや微小な磁力を感知するホール素子などのセンサアンプ部に利用されることもあります．

さらに，CMOSは高耐圧であり，たとえばオンセミコンダクタ社が製造/出荷しているMC14xxxファミリは，動作電圧が3～18Vです．この広い動作範囲は，高耐圧のメタルゲートで構成されたCMOS FETであるからこそ実現できるものであり，TTLデバイスではとても真似できません．

● LVCMOSとは

LVCMOSはLVTTLと同様に，低電圧駆動CMOSです．動作，先ほど紹介したCMOS最大の特徴と同じく，信号電圧が電源電圧まで振幅するというもので，駆動電圧の違いがあるだけです．

LVTTLの場合と同じように，駆動電圧によって「LVCMOS3.3」や「LVCMOS2.5」，「LVCMOS1.8」と明記されているデータシートもあります．

表3に，低電圧版ハイスピードCMOSデバイスであるMC74VHC245デバイスのLVCMOS特性パラメータを示します．LVTTL 3.3の場合はV_{OH}の電圧が，最小2.4V，最大値は規定なし(＝V_{CCIO}電圧と考える)でしたが，LVCMOS 3.3やLVCMOS 2.5の場合では，V_{OH}電圧はほぼV_{CCIO}と同じであることがわかります．同様に，V_{OL}の電位もほぼグラウンドレベルと，I/Oパッドに対する印加電源電圧をフルに振幅していることがわかります．

入力電圧側は駆動電圧V_{DD}に依存しますが，V_{IL}，V_{IH}ともに，V_{DD}電圧の70%以上/30%以下さえ保証できれば，直接受け付け

ることができます.

● CMOSの入出力回路

これらの特性は，CMOSデバイスの入出力I/Oパッド部分の回路構成によるものです.

この**図4**は，もっとも基本的な2入力NANDゲートの回路ですが，出力段がPチャネルMOS-FETとNチャネルMOS-FETのトーテムポール構成になっています.

PNPトランジスタとNPNトランジスタで構成されたLVTTLの出力段回路(**図3**)と似ていますが，トランジスタとは違ってMOS-FETなので，ONにすればドレインもソースも同一電位になります．これにより，上側の出力MOS-FETがONになれば(この場合は下側の出力MOS-FETはOFF)，出力ピンにはV_{DD}電圧にかぎりなく近い値になります.

逆に，下側のMOS-FETがONになれば，上側のMOS-FETはOFFになってグラウンドと接続されるので，出力ピンの電位は限りなくグラウンドレベルに近い値になります.

入力ピン近辺の四つのFET以降にあるP-MOS/N-MOSのトーテムポールゲートは，インバータゲートです.

この段数が奇数段であれば，初段の入力段がインバータ出力なので，結果として正論理出力＝ANDゲートになり，偶数段であれば負論理出力＝NANDゲートというわけです.

また，**図5**は2入力NORゲートの回路図です．**図4**と見比べてどうでしょうか．回路を構成する部品点数は変わらずに，初段の入力ピン周辺に配置されたMOS-FETの並び方が変わっただけだと思いませんか.

じつは，MOS-FETで構成されたNANDゲートとNORゲートの違いというのは，そのものずばり，入力段におけるMOS-FETの並び方の違いだけなのです.

初段のQ1～Q4で示される四つのFETのみで，NANDゲートなのかNORゲートなのかが決まり，後段のQ5-FET以降は出力時のインバータ回路によるバッファ回路です．こちらもNANDゲートと同じように，奇数段と偶数段でNORゲートとORゲートが切り替えができます.

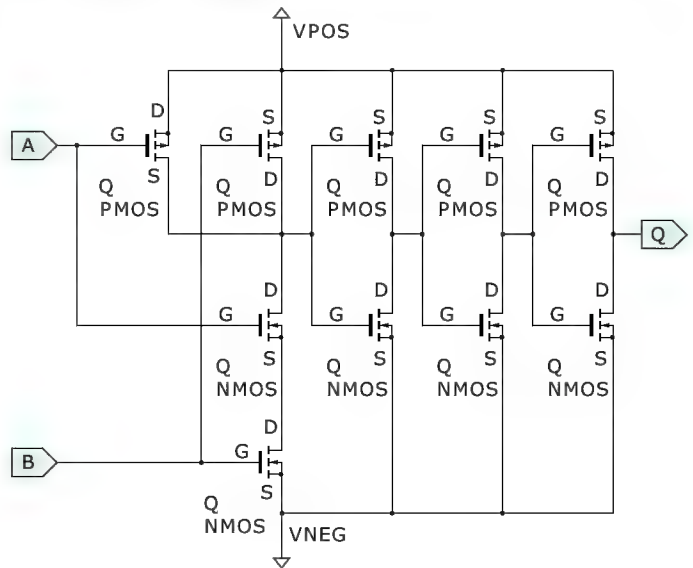
● CMOSデバイスとLVCMOSデバイスの接続

TTLとLVTTLの接続では，マージンを考えると若干不安が残るという話をしました.

これは，5.0V系TTLのV_{IH}＝最小2.0Vに対して，3.3V系LVTTLのV_{OH}も同一の2.0Vであるため，たとえばインピーダンスのマッチングや反射を防ぐためにデバイスの間にダンピング抵抗などを入れた場合には，どうしても若干の電圧ドロップが生じてしまい，マージンが取れないことを想定しています.

このような場合でも，LVCMOSであれば，“H”レベル出力電圧V_{OH}はほぼV_{DD}電圧まで振幅するので，“H”レベル側は充分にマージンが取れます．“L”レベル出力電圧V_{OL}も，グラウンドレベル付近まで落ちるので，こちらも問題なくインターフェースが取れます.

さらに，2.5V駆動でも問題なくインターフェースできる点に注目してください.

〔図4〕2入力NANDの回路

〔図5〕2入力NORの回路

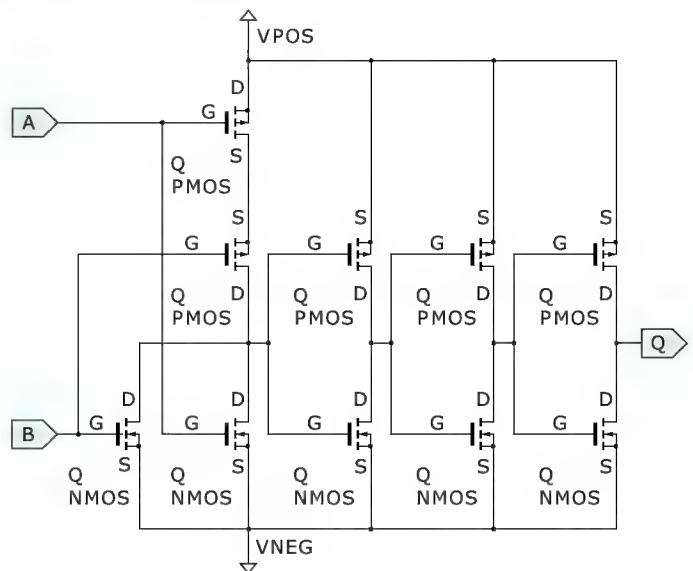
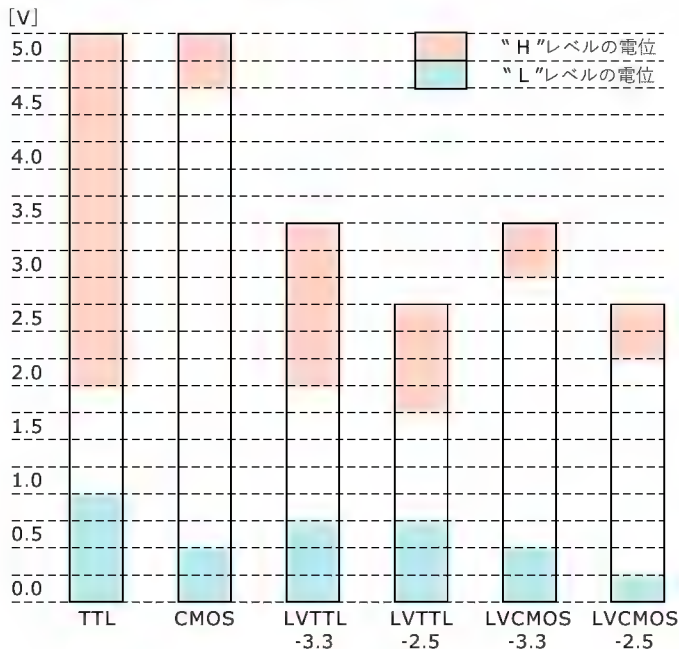



図6は，各TTL/LVTTLインターフェースとCMOSインターフェースの入出力電圧をグラフに表したものです．なお，この図のCMOSデバイスは，**表3**で紹介したMC74VHC245のものです．このデバイスは通常はV_{DD}=5.0Vで動作させますが，DC特性としては2.0～5.0Vの間であり，スペックシート上でも3.3V駆動の場合の電気的特性も明記されています.

つまり，現在の主流である低電圧駆動インターフェースを使ううえで，これらのLVCMOSタイプデバイスを用いて，LVCMOS2.5やLVCMOS3.3インターフェースを採用しておけば，同一インターフェースはもちろんのこと，さらに5V-TTLやLVTTL3.3，そしてLVTTL2.5インターフェースのすべてに対

〔図6〕LVTTLとLVCMOSの電圧比較

応することができます.

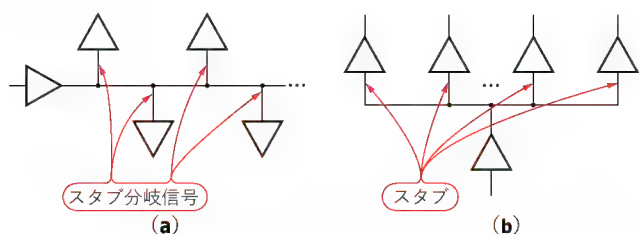
以上，LVCMOSさえあれば何とかなるという主旨で解説しましたが，じつのところLVCMOSでは昨今のプロセッサやメモリ周辺回路に見られる，100MHzを超えるようなクロックでは，信号伝達が厳しくなりつつあります．詳細については，**コラム2**を参照してください.

これに対処すべく考えられたのが，次に説明するSSTLやHSTLです.

SSTL

● 高性能FPGAでより身近になったインターフェース

シングルエンドインターフェースの中でも，いまもっともホットな信号インターフェースが，このSSTLインターフェースではないでしょうか．SSTLインターフェースは，後述するHSTLインターフェースとともに，高速なメモリインターフェース用に広く利用されています.

SSTLインターフェースは，パソコンやグラフィクスカードの主記憶メモリで一般的なDDRメモリで使われています．さらに，このSSTLインターフェースはDDRメモリだけでなく，大規模/高速FPGA間の接続インターフェースに採用する例も少なくありません.

FPGAでは，ASICやゲートアレイよりも豊富なI/Oインターフェースオプションを用意したものが多く発売されてきました．LVTTLやLVCMOSはあたりまえですが，SSTLや後述するHSTLはほぼ標準的なI/Oオプションと化しており，高性能FPGAを使った場合には容易にSSTLインターフェースを選択できます.

● SSTLとは

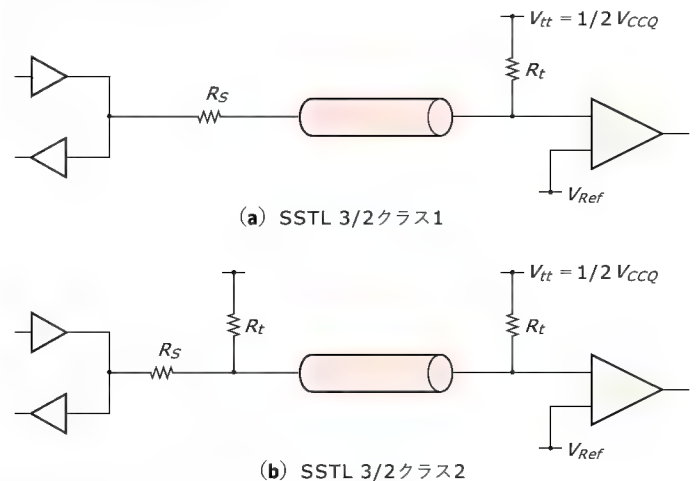
SSTLとは，前述のとおりStub-Series-Terminated-Logicの頭文字をとったものです．日本語にすると「分岐に対して直列のターミネーションを挿入した伝送線路インターフェース」とでもなるでしょうか？　日本語に訳すと，何やら呪文のごとく怪しげな言葉の羅列になってしまいますが，要はスタブに直列抵抗を挿入した伝送線路ということです．スタブは，伝送系路上から分岐した信号線の部分を示します(**図7**).

この信号インターフェースは，EIAJ/JEDEC規格に正式に登録されているものであり，I/O電源電圧により，3.3VであればSSTL-3，2.5VであればSSTL-2，そして1.8VであればSSTL-18(現在規格策定中)という使い分けがなされます．そして信号経路の形状と直列に挿入されるターミネーション抵抗の位置や値により二つのクラスがあり，SSTL-2クラス1などの呼び方がなされます(**図8**).

● V_{ref}と比較することでレベル判定

SSTL(や後述するHSTL)インターフェースが，LVTTLやLVCMOSと大きく異なる点は，既存のLVTTLやLVCMOSがグラウンドからの信号線の電位によって信号レベルを決めていたことに対して，これらのインターフェースでは，V_{ref}(リファレンス：基準)電圧と信号線上の電位を比較することによって

〔図7〕スタブ

〔図8〕SSTLのクラス

信号レベルを決めているということです．

SSTL の場合には，I/O 電源電圧の 50%（または 45%）の電圧を V_{ref} として，この電圧よりも高いか低いかで，“H”レベルか“L”レベルかを判定します．そして，このときの“H”/“L”レベルおのおのの電位差は 800mV 程度と，いままで紹介したインターフェースよりも低くなっています．

また，SSTL/HSTL では，さらにこの V_{ref} と信号線を抵抗で終端（ターミネーション）します．

もちろん，LVTTL や LVCMOS のときのように，グラウンド電位（通常は 0V）と比較することは何ら悪いわけではありませんが，ターミネーションが必須である伝送線路を想定した場合，正の“H”レベルと正の“L”レベルでターミネーションするよりも，電圧が＋/－逆ではあるが，同一の電位がとれる「中間電位」でターミネーションするほうが，その経路に流れる電流も一定になります．

たとえば，“H”レベル 2.0V，“L”レベル 0.8V の LVTTL を 50Ω でターミネーションしたとき，“H”レベル時には 40mA，“L”レベル時には 16mA の電流が流れます．しかし，中心電圧の 1.4V を中心にしてターミネーションすれば，“H”/“L”レベルともに 12mA の電流ですみます．振幅も 1200mVp-p となり，立ち上がり/立ち下がり時間の短縮にも貢献できます．

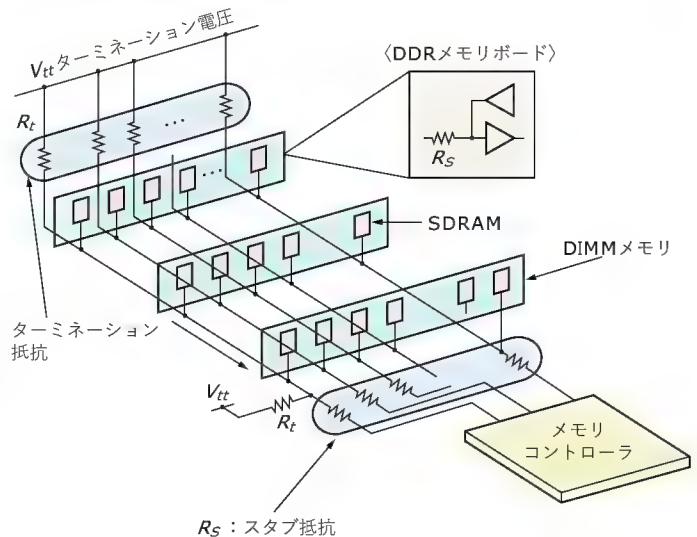
SSTL2 を例にすると，中心電圧＝V_{tt}（ターミネーション電圧）は 1.25V（I/O 電圧の 50%），SSTL インターフェースの受信側は V_{ref} 電圧とターミネーションされた信号電圧を比較して“H”か“L”かを判定しています．このことから，LVTTL/LVCMOS などと同一スルーレートであれば，t_R/t_F はこちらのほうが小さくなるということで，さらなる高速化に対応できることになります．

● DDR-SDRAM にみる SSTL の動作

ところで，昨今流行の DDR-SDRAM メモリは 2.5V 系電源電圧で駆動します．そのため DDR-SDRAM とのインターフェースに使われている SSTL インターフェースは，I/O 電源電圧 2.5V の SSTL2 クラス 1，またはクラス 2 ですが，本章では SSTL2 クラス 2 を参考に解説します．

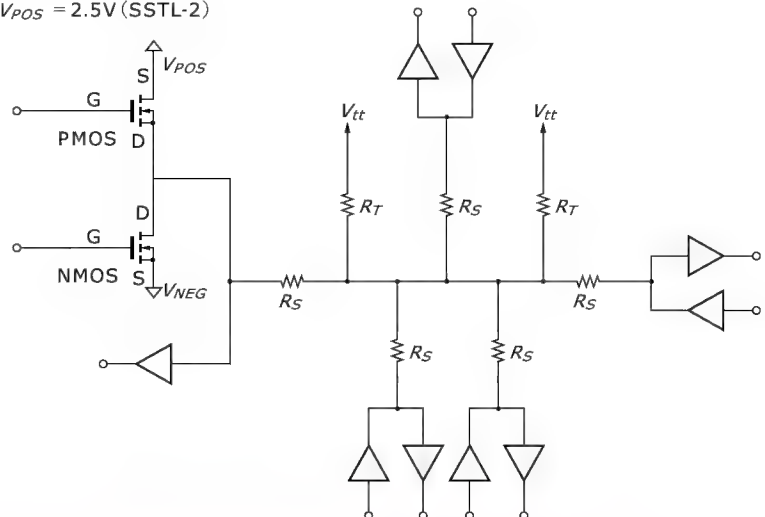
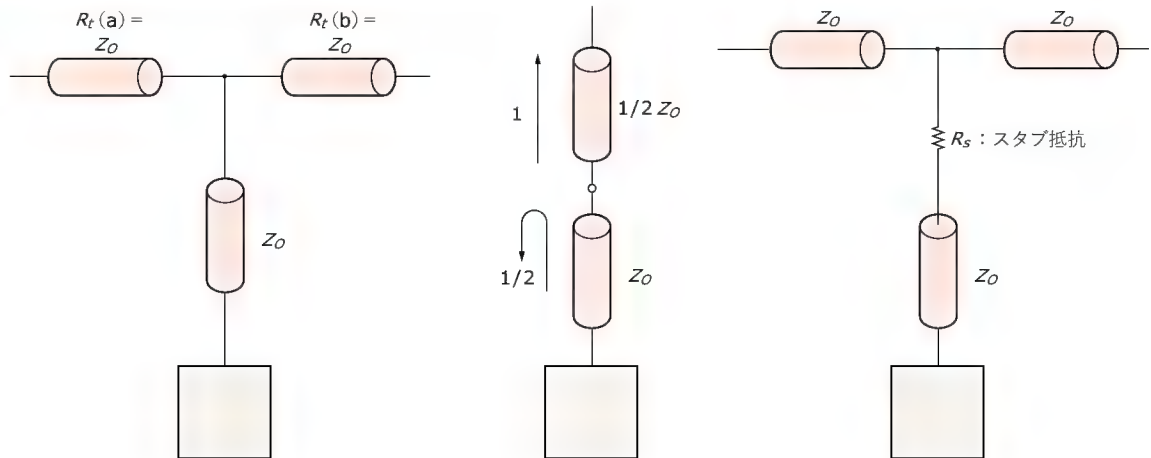
図 9 は，DDR-SDRAM を採用したシステム設計例としてのパソコンをイメージした図です．SSTL はスタブ（分岐点を複数もつ配電線路用）に取り扱うことができる信号インターフェースです．パソコンのメモリまわりを考えてほしいのですが，ユーザーは 1 枚単位で DDR メモリを購入して挿入することができます．

これはまさに，スタブがどんどん増えていくことそのものです（**図 10**）が，このような設計を想定した場合に，正しく信号伝送ができるように考えられているのが SSTL です．

図 11 は，SSTL の動作を示したものです．スタブが二つあることを想定してみましょう．ドライバからレシーバへの配電線路では，50Ω の基板固有の特性インピーダンスがあり，さらにスタブ点（A）の先には，おのおの 50Ω の抵抗が接続された T 分岐の回路が存在します．この抵抗は，特性インピーダンスとのマッチングを取るためのターミネーション抵抗です（**図 10** 中の R_t 抵抗）．すでに述べたとおり，SSTL の場合には，この抵抗の先は V_{tt} に接続されており，これは V_{ref} と同一の電位になります．

伝送線路の特性インピーダンスとマッチングを取るための R_t は終端抵抗ですが，R_t(a) と R_t(b) はともに V_{tt} に終端されているので，この図をさらに変形していくと，スタブからみてその先の終端抵抗は $1/2 \times R_t$ になります．

〔図 9〕DDR-SDRAM 搭載マザーボード

〔図 10〕DDR 信号経路

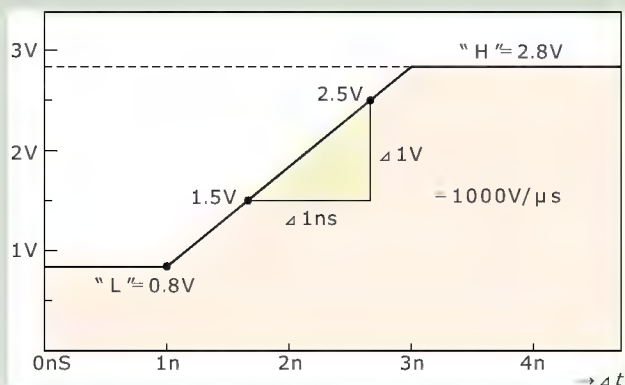
V_{POS} = 3.3V（SSTL-3）
V_{POS} = 2.5V（SSTL-2）

〔図 11〕
図 9 中の R_S の意味

Column 2
スルーレートについて

● LVTTL/LVCMOS の限界

ここで，かりに 1V の電位を 1ns で出力できるドライバがあるとしましょう．アナログ OP アンプ回路の代表的なパラメータであるスルーレートで，このドライバの性能を μs 単位の遷移値に換算すると 1000V/μs という数字になります．この値は，「高速 OP アンプ」と呼ばれる部類のもつ性能であり，相当優秀な出力アンプです(**図 C**)．

ここで 2.5V 駆動の LVCMOS を考えた場合，2.5V の 10%が“L”レベル，2.5V の 90%が“H”レベルというのが一般的なレベルの判定値になります．つまり，2.5V の 80%の範囲 = 2.00Vp-p が，LVCMOS ドライバが出力を変化させる最低の範囲の電位です．

このため，2.00V ÷ 1V/ns = 2.00ns という値が導き出され，容量性負荷や抵抗などの障害がいっさいない状況でも，レベルの遷移に 2.00ns 秒の時間が必要であるということになります．

これに IC などが接続されれば，ピンの入力容量(ファンアウト)負荷 C_L(ロードキャパシタと呼ぶ)が追加され，さらに抵抗が間に入れば，配線やピンの C_L 値によりフィルタが形成され，さらに遅くなります．

負荷のない理想状態のみを考えても，1 クロックを送るのに“L”レベルから“H”レベル，そして“H”レベルから“L”レベルへの変化で 4ns を超えます．結果として，これだけでも 250MHz より高速なスイッチングはできないわけす(**図 D**)．

現実的な話として，3.3V 系 LVTTL や LVCMOS では，信号の伝送速度はおおむね 133MHz ぐらいが限界ではないかと筆者は考えています．

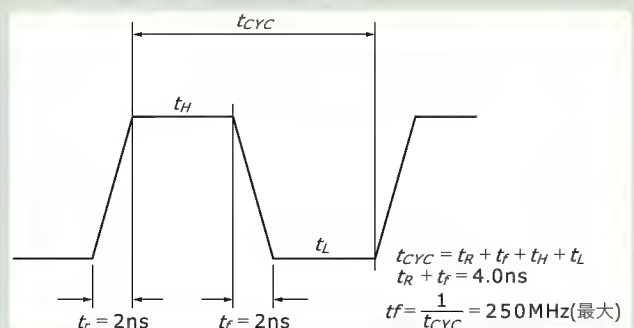
● もっと狭い電圧範囲なら……

スルーレートが同じ能力だとしても，変化させる電圧の範囲をもっと狭くできれば，もっと高速にスイッチングさせることが可能になるはずです．これが，SSTL や HSTL のアイデアです．

たとえば，SSTL インターフェース(一般的な DDR-SDRAM では 2.5V の SSTL2 クラス 1/2 を採用)では，電源電圧やグラウンドとは別に基準電圧を用意し，この基準電圧と信号線の電位でレベル判定をします．この SSTL2 インターフェースを例にして考えてみましょう(**図 E**)．

ドライバの能力は，先のものと同じ 1V/ns のドライバとします．

〔図 C〕LVTTL のスルーレート

〔図 D〕伝送できる最大クロック周波数

すると，ドライバ直前のZ_o(特性インピーダンス)とR_tは一致しなくなり，ここで1/2の信号反射が発生します．これを防ぐためには，半分になってしまった終端抵抗の合成抵抗値を持ち上げて特性インピーダンスと一致させる方法がSSTLではとられています．

そこで，スタブの直前に1/2 × R_tに相当するスタブ抵抗を直列に挿入することで，特性インピーダンスとスタブ抵抗＋終端抵抗の合成抵抗値を一致させることができ，これにより反射を防ぐことができるようになります．この伝送線路に直列で挿入されるスタブ抵抗を，R_s(Stub-Resister)と呼びます．

再度DDR-SDRAMを含めた回路系を見てみましょう．各ドライバの直前には，すべてスタブ抵抗R_sが入っていることがわかります．そして，どのドライバからみても，V_{tt}に接続されているR_tは2個しか見えておらず，先に解説した模式図にあるように，R_tは結局1/2 × R_tとなることがわかると思います．特性インピーダンスはR_tと同じ値なので，そこでドライバそれぞれにR_sを入れることで，マッチングを取っているのです．

なお，レシーバ側の入力インピーダンスは非常に高く，実質電流は流れないことで，いくつR_sが挿入されていようが伝送線路には影響がないということがあらかじめ前提としてあります．

SSTLインターフェースの特徴は，V_{ref}と信号レベルを比較するので，低い振幅で伝送を行う点であることは先に述べました．

表4に，SSTLの各スペックに基づいたDCレベルの電気的特性を示します．およそ基準電圧に対して，400mV以上の電圧振幅さえあれば，レベル判定が正しく行われるという点は，LVTTL/LVCMOSのそれとは違って，非常に驚くところです．

“H”/“L”レベル合わせて800mVp-pあれば信号伝送できるので，同一スルーレートであればLVTTL/LVCMOSのおよそ2

SSTL2は2.5Vの1/2電位である1.25Vが基準電圧となります．DDR-SDRAMを考えた場合，そのほとんどが基準電圧に対して±300mVを超えるとレベル変化と認定されます．すなわち，“L”レベルは950mV，“H”レベルは1550mVの電圧となります．

その電位差は600mVとなるので，遷移時間は600mV ÷ 1V/nsで，600psであることがわかります．

LVCMOSと同様に，立ち上がり時間と立ち下がり時間を加算しても1.2nsなので，およそ800MHzぐらいの転送帯域まではいけそうな気配です．

つまり，スルーレートが同一の特性をもつドライバであっても，遷移時間を小さくすることで高速なデータ信号を行えるわけです．

現実的な話では，現在DDR-SDRAMでバスクロックが200MHzというものがパソコンの世界には広まってきており，一部ではDDR-2の533(266MHz)や600(300MHz)というものもあります．

ドライバの性能と小振幅電圧インターフェースの進化により，シングルエンドでも500MHzぐらいまではまだまだ軽く進化しそうな雰囲気です．

● ノイズ耐性

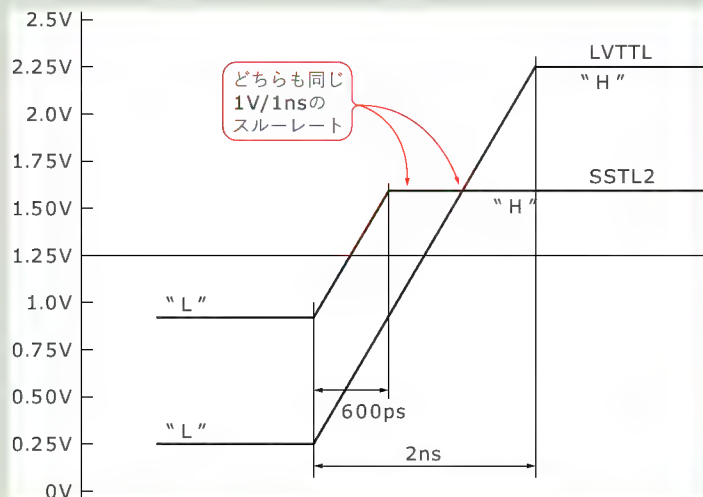
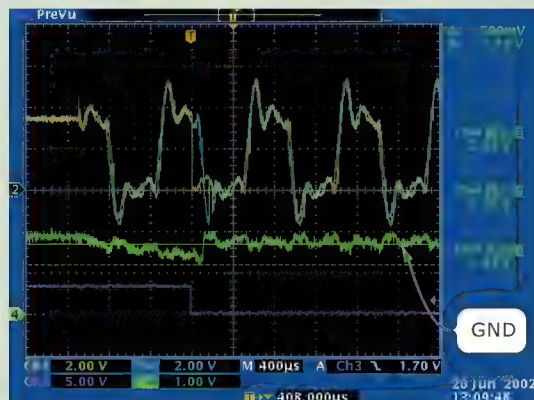
しかし，今度は振幅が小さいために，たとえば平走する隣り合った信号ラインからノイズが飛んで，信号の電位がずれてしまった場合に対する余裕度，つまりノイズマージンは小さくなります．ノイズマージンが小さいと，とくにグラウンドレベルが多少不安定であった場合には誤った信号とみなされてしまうかもしれません．

写真Aは，128ビットデータバスをドライブした際にグラウンドレベルがずれてしまう現象を観測した波形です．これが高速でシステムを動作させると無視できなくなる「グラウンドバウンス」です．

とはいえ，このV_{ref}がグラウンドバウンス発生時と同様にふらついていたのでは意味がありません．SSTLやHSTLでは，このV_{ref}の安定度も重要な設計留意点となります．

〔図E〕LVTTLとSSTLのスルーレートの比較

〔写真A〕グラウンドバウンスのようす[1)]

New Products ── 4基のSERDESチャネル搭載フィールドプログラマブルシステムチップ
ラティスセミコンダクター社は，フィールドプログラマブルシステムチップ(FPSC)の新製品として，それぞれが0.6～3.7Gbpsで動作する4基のSERDESチャネルを搭載し，エンベデッド8b/10bエンコーディング機能などを備えた「ORT42G5」を発売した．

〔表4〕SSTLの特性表

SSTL	SSTL3		SSTL2		単位
	クラス1	クラス2	クラス1	クラス2	
V_{DDQ}：駆動電圧	3.3V		2.5V		V
V_{tt}：ターミネーション抵抗	V_{DDQ} × 0.45 (= 1.5)		V_{DDQ} × 0.5 (= 1.25)		V
V_{ref}：リファレンス電圧	V_{DDQ} × 0.45 (= 1.5)		V_{DDQ} × 0.5 (= 1.25)		V
Hレベル電位(DC最小)	V_{ref} + 0.2		V_{ref} + 0.18		V
Lレベル電位(DC最大)	V_{ref} − 0.2		V_{ref} − 0.18		V
R_t：ターミネーション抵抗	50	25	50	25	Ω
R_s：シリーズ抵抗	25	25	25	25	Ω
I_{OH}(計算値)	8	16	7.6	15.2	mA
I_{OL}(計算値)	− 8	− 16	− 7.6	− 15.2	mA

倍の信号伝送帯域が期待できます．

たとえば通常，シングルエンドインターフェースにおいて，t_R/t_F の時間はサイクルタイム全体の5%以下に抑えることがほとんどです．このため，かりに4V/nsのスルーレートであれば，LVCMOS2.5の場合には各0.625nsの t_R/t_F なので，1サイクルは25ns(つまり50MHz程度)となります．それに対して，800mVp-p振幅のSSTL2では各0.2nsの t_R/t_F であるため，1サイクルは8ns(つまり125MHz)となります．

なお，執筆時点では，SSTL 1.8についてはEIA/JEDEC規格でまだ規定されたものはありませんが，SSTL2の回路電圧のVDDQを1.8Vにする以外は，V_{tt} や V_{ref} の概念や電圧振幅は同様なものです．

また，信号振幅が小さいことで外部に影響を与えるノイズ発生量，いわゆるEMIが小さくなり，隣りあう信号線間との結合容量によって発生するクロストークの影響も小さくなります．

● SSTLの応用

SSTL信号の応用としては，単にLVTTL/LVCMOSの置き換えというだけでなく，DDR-SDRAMに代表されるようにダブルデータレート，つまり一つの信号線で，クロックの立ち上がりと立ち下がりエッジを利用して2倍のデータ転送を行う方法があります．

また，昨今のFPGAではI/Oインターフェースオプションが豊富であり，I/O電圧が2.5Vの場合にはLVTTL，LVCMOSと同時にSSTL2を同一のI/OパッドやI/Oバンクで使うようなことも可能です(ただし，SSTLを利用する場合には別途 V_{ref} 基準電圧を供給する必要がある)．

これによって，大容量の転送を複数のFPGA間で行わなければならないような場合には，SSTLインターフェースは，V_{ref} 基準電圧電源とターミネーション抵抗の追加によって利用できるという利便性があります．

なお，ダブルデータレート転送を行うDDR-SDRAMメモリの動作についての詳細は，姉妹誌『デザイン ウェーブ マガジン』2003年3月号を参考にしてください．

HSTL

● HSTLとは

HSTLインターフェースは，1.5V電源電圧で駆動する高速シングルエンド信号インターフェースで，ラムバス社のラムバスメモリを駆動するインターフェースとしても知られています．また，PowerPCのL3キャッシュに接続される超高速のキャッシュメモリ用SRAMにも使われています．

電源電圧が1.5V，そして信号レベルはSSTLインターフェースよりもさらに低い電圧レベルであるため，LVTTL/LVCMOSやSSTLに比べて次の点で効果が期待できます．

- リファレンス電圧と比較した電圧振幅により信号レベルを確定するので，SSTLと同様にドライバの供給電圧には依存しない．つまり，多少の電源変動には強い
- V_{ref} とドライバの電圧はユーザー側である程度融通が利く
- 信号振幅がSSTLよりもさらに低いので，外乱ノイズの発生が少なく消費電力も小さい

HSTLでも，SSTLと同様に，信号経路の構成により複数のクラスがあります．

▶ HSTLクラス1

V_{DDQ}(ドライバ/レシーバ駆動電源電圧)に1.5V，そして V_{ref} には V_{DDQ} の1/2である0.75Vをレシーバのリファレンスピンに供給する方式です．

伝送線路は，通常50Ωの終端抵抗で V_{ref} に終端されます．これは，他のクラスでも同じです．信号の振幅はリファレンス電圧を中心として±400mVなので，R_t=50Ωの場合には"L"レベル/"H"レベルともに8mAの電流が流れます．これにより，1ピンあたり3.2mWの電力が抵抗で消費されます．

信号経路は，SSTLクラス1とまったく同一です．

▶ HSTLクラス2

HSTLクラス1の例と同様に，こちらもSSTLクラス2と同一の信号経路です．ドライバのピン近辺とレシーバのピン近辺にターミネーション抵抗を介してリファレンス電圧に接続されます．ターミネーション抵抗は，50Ωが一般的です．

▶ HSTLクラス3

HSTLクラス3では，ターミネーション電圧とリファレンス電圧がクラス1，クラス2と異なります．

信号経路はHSTLクラス1と同一ですが，V_{tt} はドライバ/レシーバの駆動電源電圧である V_{DDQ} = 1.5Vと同一になります．ターミネーション抵抗は通常50Ωであり，V_{DDQ} に接続されます．そして，V_{ref} は V_{DDQ} の60%にあたる，0.9Vになります．

V_{tt} がドライバ/レシーバの駆動電圧であるということは，"H"レベルの信号はほぼ V_{DDQ} そのものになることが想像できます．それに対して，"L"レベルは最大で0.4Vと規定されています．実際には，V_{DDQ} − V_{tt} の0.6Vが振幅電圧として，同様に"L"レベルも V_{tt} − 0.6Vとなるので，0.3Vぐらいまで下がります．

〔表5〕HSTLの特性表

HSTL	クラス1	クラス2	クラス3	クラス4	単位
V_{tt}：ターミネーション抵抗	V_{DDQ}/2(=0.75)	V_{DDQ}/2(=0.75)	V_{DDQ}(=1.5)	V_{DDQ}(=1.5)	V
V_{ref}：リファレンス電圧	V_{DDQ}/2(=0.75)	V_{DDQ}/2(=0.75)	V_{DDQ} × 60%(=0.9)	V_{DDQ} × 60%(=0.9)	V
Hレベル電位(DC最小)	Vtt+0.4	Vtt+0.4	V_{DDQ} − 0.4	V_{DDQ} − 0.4	V
Lレベル電位(DC最大)	Vtt − 0.4	Vtt − 0.4	0.4	0.4	V
I_{OH}(計算値)	8	8	8	8	mA
I_{OL}(計算値)	− 8	− 8	− 48	− 48	mA

つまり，HSTLの規格上では0.4～1.5Vまでの1.1Vp-pという，V_{DDQ}のほぼ全域にわたって電圧振幅をします．

このクラスでは，HSTL信号規格のうえで，外乱ノイズやクロストークノイズなどのノイズマージンを考慮して，あえて低電圧振幅を期待せずに1.8VのLVCMOSのように高いノイズ耐性が必要な場合に使用されるインターフェースです．

▶ HSTLクラス4

HSTLクラス3と同じV_{tt}やV_{ref}で，信号経路の構成がHSTLクラス2と同じものです．クラス3同様に，高いノイズ耐性が期待できます．

これらのクラスによる違いをまとめたものを**表5**に示します．信号経路の構造は，すでに説明したSSTLのものとかわらず，リファレンス電圧とドライバ/レシーバに対する駆動電圧が違うだけなので，この機会に双方の規格を頭の片隅に置いておくのもよいかと思います．

GTL/GTL+/AGTL/AGTL+

このインターフェースは，Xeroxが低電圧駆動の高速なロジックインターフェースとして1991年に考案されたもので，発明者のBill Gunningの名前にちなんで，GTL(Gunning Transceiver Logic：ガンニングトランシーバロジック)と名づけられました．GTLは，数百MHzの高クロック周波数帯域におけるデータ転送が必要な通信機器で，筐体内におさめられたバックプレーンやCPUとI/O間を接続するインターフェースとして利用されています．

〔表6〕GTL/GTL+/AGTL/AGTL+の特性表

信号名	GTL	GTL＋	AGTL	AGTL＋	単位
V_{TT}ターミネーション電圧	1.20	1.50	1.25	1.50	V
V_{REF}基準電圧	0.80	1.00	0.83	1.00	V
電圧振幅	± 50	± 200	± 20	± 200	mV
終端	必須	必須	必須	必須	－

GTLはSSTLやHSTLと同様に，V_{ref}を使って信号レベルを判定する，オープンドレイン方式の信号インターフェースです．V_{TT}電圧＝1.2V，V_{ref}＝0.8Vで，SSTLクラス3や4に似ていますが，信号振幅はさらに低く，V_{ref}に対して±50mV(=100mVp-p)となっています．

そして，インテル社はGTLをベースに，Pentium Proのバスとして，クロック66MHz/8デバイスがサポートできるよう，V_{TT}=1.5V，V_{ref}=1.0V，電圧振幅を±200mVに変更したGTL+を開発しました．そして，この信号インターフェースをシステムバスの特許として成立させてPentium IIまで使用されました．

その後，インテル社はPentium IIIのシステムバスとして，GTL+の規格を元に，オープンドレインの出力ドライバに補正，つまりアシスト機能を追加したAGTL(Assist GTL)/AGTL+インターフェースを採用し，現在に至っています(**表6**)．

ディファレンシャルインターフェース編

ECL/PECL

● 古くから利用されている高速インターフェース

ECLは古くから利用されている信号インターフェースで，非常に長い歴史があります．もっとも，コンピュータの世界で「非常に長い歴史がある」と書いたとしても，たかだか20年～30年程度のことですが，筆者が生まれた1970年代前半の大型コンピュータでは，すでにこの信号インターフェースが使われていました．

執筆の関係で，当時の資料をインターネットで検索したところ，すでに1975年時点で，ECLデバイスのIC内部におけるゲート間遅延時間は500ps～700ps，チップ外側でも2nsという速度のECLデバイスが使われていたようです．

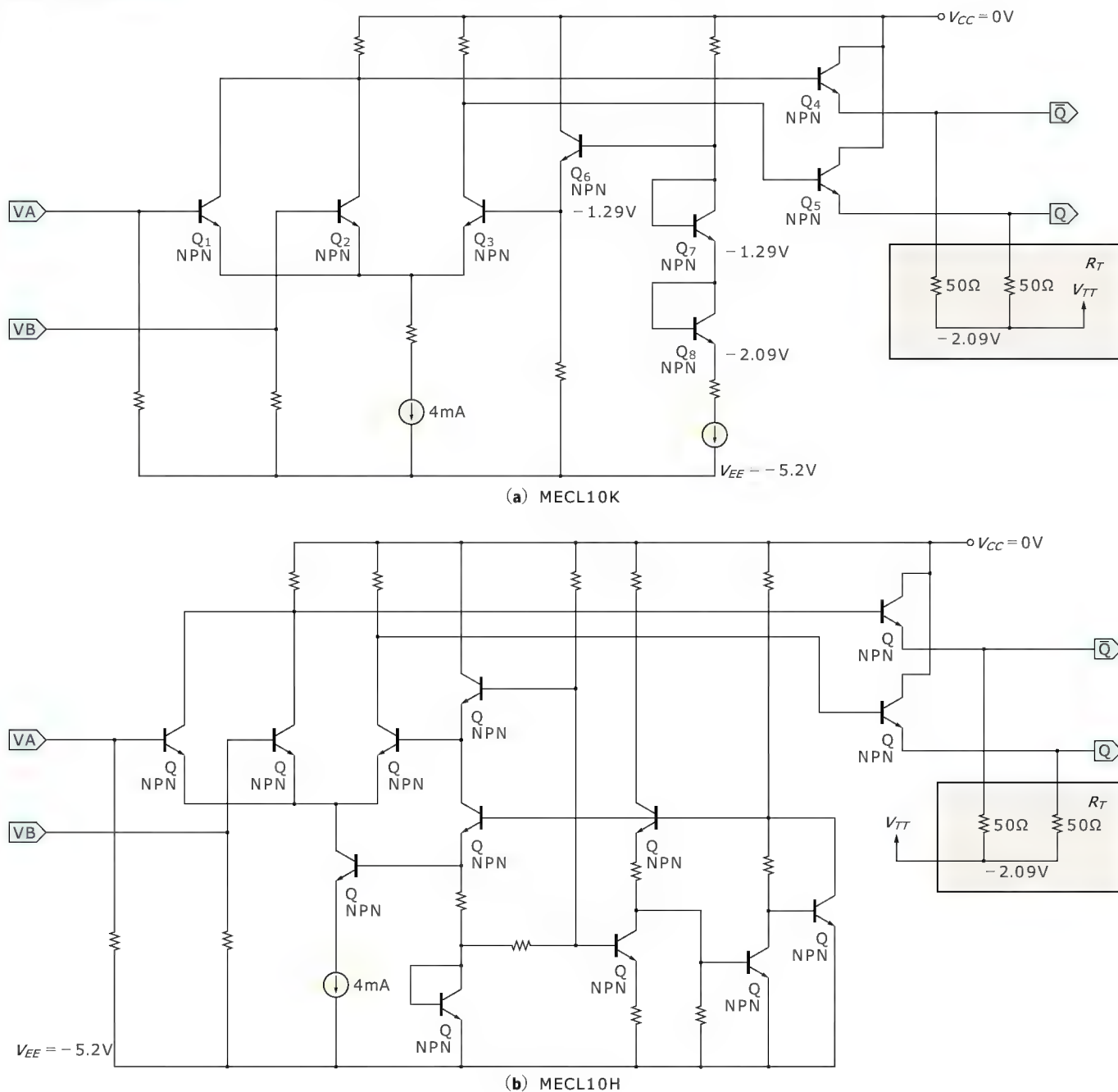
● ECLとは

ECLとは，Emitter-Couplered-Logic(エミッタ結合型論理回路)の頭文字をとったものであり，バイポーラトランジスタで回路が構成されています．ECL回路の基本は，OR/NORゲートです．TTL回路の基本がNANDゲートであることでは，基本論理が違う点に気をつけなければなりません．

図12(a)は，OR/NOR型ゲートロジックの基本回路を表したものです．この回路は，米国モトローラの開発したMECL(Motorola-ECL)製品群の中の，10Kファミリと呼ばれるもの

〔図12〕ECLの回路

(a) MECL10K

(b) MECL10H

で，ECL登場初期の回路です．型番はMC10xxxとなっています．

基本構成はトランジスタで構成された差動アンプで，ECLの基本動作はトランジスタの非飽和領域を使った差動アンプの動作そのものです[注]．

片側の差動アンプの入力側には，コレクタとエミッタが結合された複数のトランジスタが接続され，そのベースは外側に出力されています．これがECLの入力ピンです．差動アンプの片側はコレクタに印加される電圧から1.29V下がった定電圧が接続されます．

回路図右端は，この−1.29Vを生成するためのレギュレータ回路です．10Kファミリは，抵抗と2組のダイオードにより電

注：TTLやCMOSは電源電圧フルに振幅を振って飽和させた「スイッチング動作」だが，ECLではV_{BE}-I_B特性の急峻な電流カーブを有効に使った「アナログ動作」である．詳細は後述するが，十分なゲイン（増幅度）をもったトランジスタであれば，この非飽和領域を使うことで，高速なスイッチングを行える．

Column 3

消費電力についての考察

低電圧で駆動する時代になると，消費電力のことも考慮しておかねばなりません．LVTTLやLVCMOSインターフェースでは，どのようなときに電力を消費するのでしょうか．

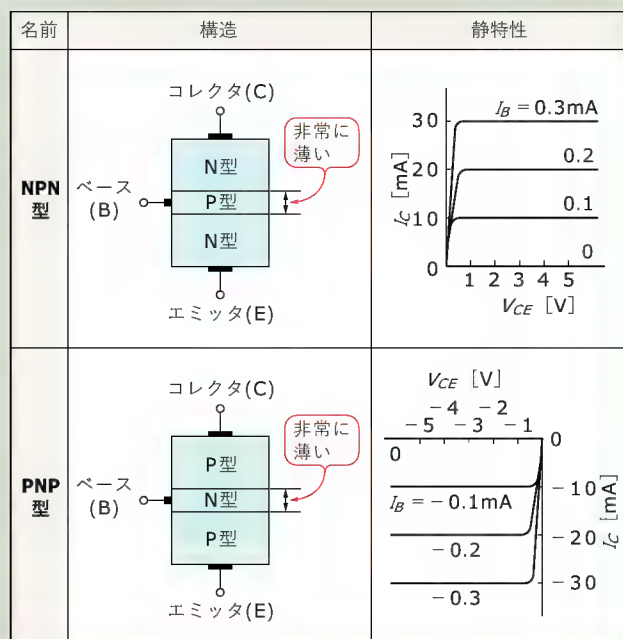
LVTTLを構成する回路を，基本構成を忠実に考えてトランジスタで構成されたものと想定して考えてみます．

トランジスタのON状態，またはOFF状態を制御するためには微弱ながらも電流がトランジスタのベース-エミッタ間に電流I_bが流れなければなりません．そして，この電流が流れることで，コレクタ-エミッタ間に電流I_{ce}が流れるか流れないかが定まります(**図F**)．トランジスタがOFF時であればI_bは流れませんが，ON時であればI_bは必ず必要であり，その電流量は数μA～数十μAです．

次は，LVCMOSデバイスを構成する回路を，MOS-FETで構成されたものを考えてみましょう．MOS-FETの場合には，すべてエンハンスメント型のFETで動作の説明ができます．MOS-FETの動作は，ゲート-ソース間の電圧V_{gs}が印加されることで，ドレイン-ソース間に電流I_{dds}が流れます．Nチャネル MOS-FETの場合，V_{gs}の電圧が高ければ高いほどI_{dds}が流れますし，逆にV_{gs}が0VであればI_{dds}は流れません．これにより，ドレイン-ソース間でON/OFFを切り替えます．Pチャネル MOS-FETは，この動作が逆です(**図G**)．

そして，トランジスタとは違ってドレイン-ソース間がONのときに必要な，ゲート-ソース間に流れる電流＝I_{gs}はnA，もしくはpAオーダ，場合によってはfA(フェムトアンペア)オーダまで小さいものもあります．

〔図F〕トランジスタの動作

名前	構造	静特性
NPN型	コレクタ(C) ベース(B) N型 P型 N型 非常に薄い エミッタ(E)	I_C [mA] 0, 10, 20, 30 I_B = 0.3mA, 0.2, 0.1, 0 V_{CE} [V] 1 2 3 4 5
PNP型	コレクタ(C) ベース(B) P型 N型 P型 非常に薄い エミッタ(E)	V_{CE} [V] −5 −4 −3 −2 −1 0 I_B = −0.1mA, −0.2, −0.3 I_C [mA] −10, −20, −30

スイッチとして素子を利用した場合，MOS-FETで必要な電流はトランジスタにくらべて，1/1000以下の微小なものです(あくまで，個別半導体で構成した場合を想定)．

よって，トランジスタで構成されたLVTTLインターフェースの場合には，“H”か“L”かのいずれかの一定の信号レベルに保つため，いくつかのトランジスタを駆動するわけですが，この際に必ず電流が流れ続けるわけです．

それに対してCMOSデバイスは，論理の確定時点ではほとんど電流は流れません．よって，トランジスタ構成のLVTTLと違ってLVCMOSは消費電力が非常に小さいのです．

とはいえ，“H”→“L”または“L”→“H”にレベルが変更する動作状態はどうでしょう．結論をいうと，LVTTLもLVCMOSも，この状態に電力消費をします．

このほかにも，トランジスタやFET素子はある程度の容量性負荷をもっており，前段の素子から後段の素子をドライブするためにこのような容量性負荷に対するチャージのための電流が必要になることも見逃せません．

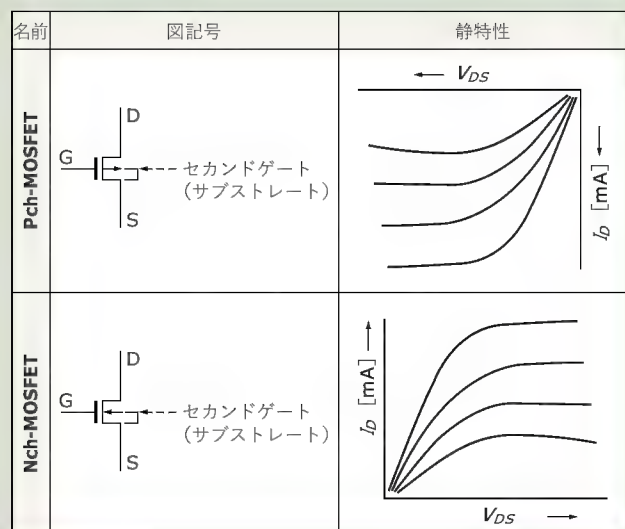
これらの事情から，低消費電力をめざしているデバイス，とくにディープサブミクロンプロセス世代では，

(1) トランジスタのスイッチング速度を上げる
(2) スレッショルド電圧に対する感度を上げる
(3) スイッチングデバイスの負荷容量を小さくする

などの対策を施します．

(1)と(2)を組み合わせることで，スイッチ切り替えの瞬間の貫通電流を減らせます．また，とくにゲインを上げるとミラー効果で見かけ上の容量性負荷が増えるため，ゲインを抑えるためにシリコンに金を混入(ゴールドドーピング)してゲインを下げたり拡散膜を極限まで薄くするといった方法を採ります．

以上のような努力は，各半導体メーカーではあたりまえのように行っており，今日の高性能な半導体製品があるのです．

〔図G〕FETの動作

〔表7〕**ECL特性表**

パラメータ	ECL-10K/10H			ECL-10E/100E			LVPECL-MC100LVEL			単位
	最小	標準	最大	最小	標準	最大	最小	標準	最大	
V_{CC}：上側電源電圧	−0.05	0.00	0.05	−0.05	0.00	0.05	3.14	3.30	3.47	V
V_{EE}：下側電源電圧	−5.46	−5.20	−4.94	−5.25	−5.00	−4.75	−0.05	0.00	0.05	V
V_{tt}：ターミネーション電圧	−2.19	−2.09	−1.99	−2.19	−2.09	−1.99	1.14	1.30	1.47	V
V_{IH}：Hレベル入力電圧	−0.96		−0.81	−1.16		−0.88	2.14		2.42	V
V_{IL}：Lレベル入力電圧	−1.85		−1.65	−1.81		−1.47	1.49		1.85	V
I_{IH}：Hレベル入力時リーク電流	50.00					150.00			150.00	μA
V_{OH}：Hレベル出力電圧	−0.96		−0.81	−1.03	−0.95	−0.88	2.27	2.35	2.42	V
V_{OL}：Lレベル出力電圧	−1.85		−1.65	−1.81	−1.75	−1.62	1.49	1.60	1.68	V

注意：ECL-10E/100EはV_{CC}=5.0V，V_{EE}=0Vで，5V-PECLとしても使用可能．その場合のV_{tt}は3.00V/50Ωターミネーションを原則とする．

流値を決め，そしてV_{EE}から1.29Vの電圧を作るためのエミッタ抵抗値が決まるような，簡易的なレギュレータ回路になっています．

そのため，電圧変動や温度変化による1.29Vの定電圧特性が安定しないという問題があるため，こののちに10Hや100Hなどの新しい回路構成をもつECLデバイスに切り替わっていきました〔**図12**(**b**)〕．

なお，MECL-10Kファミリと区別するために，型番はMECL-10HxxxやMECL-100Hxxxとなっています．

● PECL

ところで，ECLデバイスを使う場合には，通常よく使う＋5V/0Vのような正電圧ではなく，0V/−5.2V(ECL-10K)や0V/−4.8V(ECL-10H/100H)のような負電圧が必要です．

そのため，一般的に使用される正電圧駆動のTTL/CMOSデバイスとの混在設計時において，高速とはいえ負電圧が必要なECLは敬遠されました．それを受け，後に正電圧で駆動するECLデバイスが登場しました．これがPECLデバイスです．なお，一般的にPECLというと，Positive-ECLという名前で呼ばれますが，1991年に筆者がはじめてPECLを使ったときにはPseudo-ECL，つまり疑似ECLと呼ばれていました．

ちなみに，ECLデバイスは仮に10Kファミリでも，＋5V/0VのTTLレベルで使用が可能です．レベルが5V系のTTLやCMOSと異なるだけで，ちゃんとこれらのデバイスを駆動できます．筆者は100Hファミリを無理やり5Vで駆動し，出力を高速のPNPトランジスタを使ったレベルシフタでTTLレベルまで落として使用した実績があります．出力レベルの動作点さえしっかりおさえておけば，いろいろな使い方ができるのがECLでした．

● LVPECL

そして，今日現在では，5V系正電圧で動くPECLに対して，3.3V系正電圧で動作するローボルテージ版のLVPECL製品が商品化されています．

米国モトローラでは，ECLinPS(エクリンプスと呼ぶ)ファミリという名前で出荷されており，ゲートロジック，400MHz～800MHzクラスのPLLクロックシンセサイザ，クロックディストリビュータ(クロック分配デバイス)やLVPECL→LVTTLレベルシフタなどのデバイスファミリが出荷されています．

また，このLVPECLインターフェースは，昨今の高性能FPGAでも使用できるようになっており，LVPECLは数百MHzからGHzの一歩手前程度までの伝送方式として，一般的な信号インターフェースになるのではないでしょうか．

● ECL/PECL/LVPECLの特性

表7は，ECL/PECL/LVPECLの各ファミリごとのレベルを示したものです．

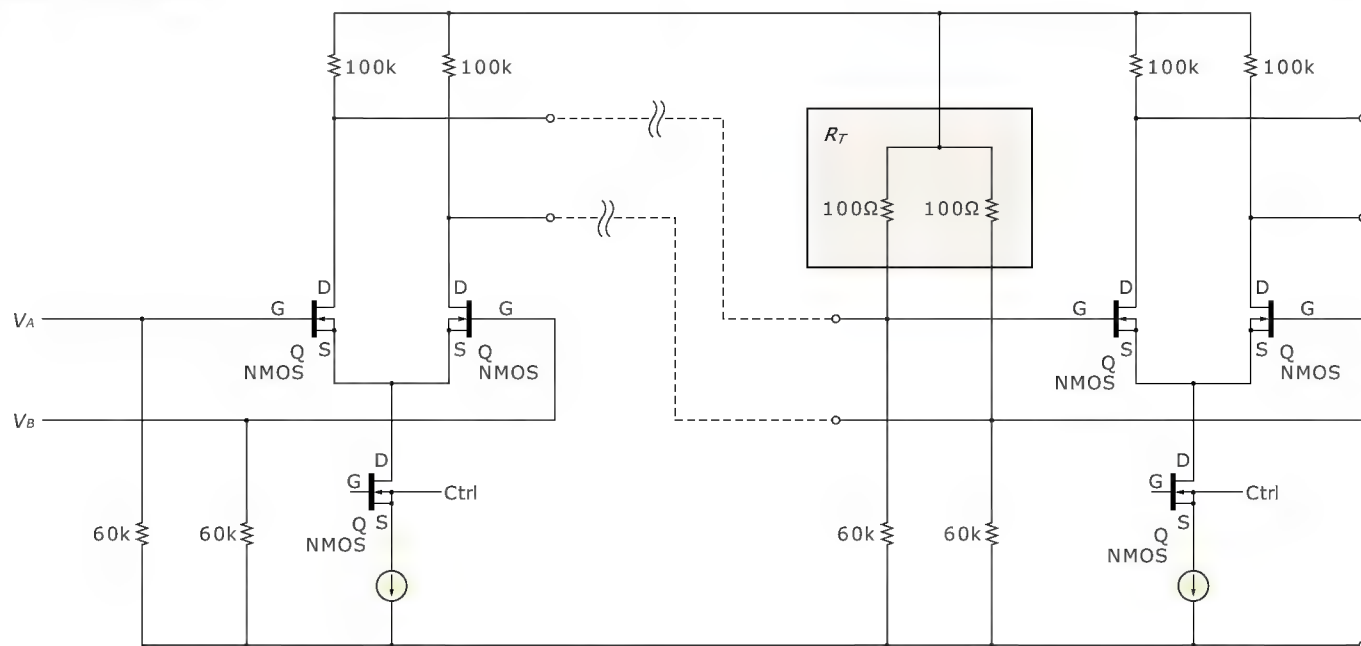
信号の振幅はおよそ800mV程度であり，レベルが反転した2本の差動信号で伝送されます．ECL系の出力段は，単純なトランジスタのエミッタ端子がそのまま外に出力された形になっていることが特徴であり，外側には必ずV_{tt}(ターミネーション電圧)との間にR_t(ターミネーション抵抗)を実装することが必要です．

このターミネーション抵抗は，一般的にECL/PECL/LVPECLともに50Ωに設定されており，伝送線路の特性インピーダンスZ_oもこのR_tと同様の50Ωでのインピーダンスコントロールが推奨されます．

ターミネーション抵抗が接地するV_{tt}は，V_{CC}電圧から2.0V低い電圧に設定されます．ECLの場合には−2.0Vが，5V駆動のPECLの場合には3.0Vが，3.3V駆動のLVPECLの場合には1.3Vがターミネーション電圧となります．これらの電圧はV_{CC}/V_{EE}電圧以外にリニアレギュレータやDC/DCコンバータなどの別電源で作成して供給しなければならない点が面倒なところです．

なお，トランジスタのエミッタが単純に外側に出力されているだけなので，ターミネーション抵抗をつけないと，電圧振幅は発生しないので注意が必要です．

また，エミッタが単純に外に出されているだけなので，ドライブ能力が足りないと思ったときには，複数のゲートを束ねてドライブ能力を稼ぐことができ，数十ものデバイスが連なったハイファンアウト(高負荷)状態を駆動する際には有効な方法です．もっとも，最近のLVPECLでは，このようなアプリケーションノートは見られなくなり，ハイファンアウト対応のバッ

〔図13〕CMLの回路

ファ（ディストリビュータ）の使用を推奨しているようです．

● 消費電力

ECL/PECL/LVPECLは，内部構成がバイポーラトランジスタで作られているために，常時電流を流しておかなければならないという仕様になっています．

最近はCMOSプロセスに移行していますが，10Kや10Hファミリはバイポーラトランジスタを差動ペアの入力段に用意しているので，トランジスタに流れ込むバイアス電流もCMOSデバイスの比ではありません．やはり，ここでも電流が流れてしまいます．

また，振幅そのものは800mVと小さいのですが，50Ωのターミネーション抵抗はTTL/CMOSのようなシングルインターフェースにおけるプルダウン抵抗と同じでV_{tt}電圧に接続されているために，“H”レベルでも“L”レベルでもこの抵抗でかなりの電力が消費されてしまいます．

これらの理由から，レベルが固定された状態でも電力が消費され，「ECLは熱い」，「ECLは大電力食いだ」とよく言われます．

実際，ECLを採用したシステムは冷却系にも気配りが必要であり，メインフレームなどの大型汎用コンピュータでは循環液体冷却型の冷却システムを装備しているものも少なくありません．

低電圧/低消費電力システムの需要が高まる現代社会においては，いささかECL/PECL/LVPECLは古いデバイスであるために，採用が敬遠されがちのようです．低電圧/低消費電力が要求される昨今では，ECL構成はたしかに厳しい状況であり，今後はCMLやLVDSに徐々に切り替わっていくと予想されます．

CML

● CMLとは

CMLとは，Current-Mode-Logic（カレントモードロジック）の頭文字をとったものです．じつは，筆者はこの執筆を行うまで，CMLはバイポーラトランジスタのコレクタをイメージした，Corrector-Mode-Logic（コレクタモードロジック）のことだと思っていました．CMLの回路構成をみると，筆者がそう思っていた理由がすぐわかると思います．

図13に，FETで構成したCML回路を示します．これからわかるように，CMLは差動アンプのコレクタ側につながるパスが出力端子となっており，次段の入力端子に接続されます．そしてCMLでは，外部でV_{CC}電圧と接続されたターミネーション抵抗が必須になります．

別の言い方をすれば，差動アンプ部がNPNバイポーラトランジスタ，またはNMOS-FETで構成されているために，“L”レベルに落とす能力はあっても“H”レベルに持ち上げる能力は基本的にないので，V_{CC}電圧に接続された抵抗が必要になるということなのです．

ただし，最近のCMLデバイスではあらかじめV_{CC}電圧と差動アンプ部の間に抵抗が入ったものもあります．これは，外側の抵抗がなくとも電流経路が確保されて出力電圧を得ることができる利点があります．ただし，この場合でも超高速帯域で利用されるCMLは一般的に伝送線路の特性インピーダンスとマッチングさせるために，内蔵された抵抗または特性インピーダンスと同等の抵抗を外部に接続します．

ここで，前項のECLの解説で紹介した回路構成〔**図12(b)**〕と

〔表8〕
第2章で紹介する光モジュールの特性表

パラメータ	OptoCube-LVDS モード			OptoCube-CMOS モード			単 位
	最 小	標 準	最 大	最 小	標 準	最 大	
V_{CC}：上側電源電圧		3.30			3.30		V
V_{EE}：下側電源電圧		0.00			0.00		V
V_{cm}：コモン電圧	$V_{EE}+0.85$	1.20	3.50				V
V_{diff}：差動電圧	0.20	0.80	1.90				V
入力容量	1.00	1.30	1.60	1.00	1.30	1.60	pF
V_{OH}：Hレベル出力電圧				2.00		$V_{CC}+0.3$	V
V_{OL}：Lレベル出力電圧				−0.30		0.80	V

見比べてみましょう．出力トランジスタのQ_4/Q_5のベース入力につながる差動アンプのコレクタ端子がCMLの出力ピンであり，Q_4/Q_5トランジスタ自身は受け側の入力差動アンプそのものであることが見て取れます．つまり，CMLはV_{CC}から差動アンプのコレクタにつながる電流経路が外側に依存したものであり，ECLの回路と比較しながら動作を理解することができます．

もっとも，ECLの場合は電流出力型であり，CMLの場合には電流引き込み型である点が違いますが，その違いは差動アンプの先に出力トランジスタがあるかないかによる違いからきます．

● RS-422などでも使われている

さて，CMLもECLと同様にたいへん古くから使われています．平衡式の差動インターフェースであるRS-422は，±20mAの電流駆動方式を採用したCMLです．RS-422では信号レベルとして電圧をやりとりしているのではなく，ドライバ側デバイス内の差動アンプによる引き込み電流の違いによりレベル判定を行い，レシーバ側の終端抵抗によりレシーバ側の入力ピン電位が決定されます．

RS-422の場合，終端抵抗は100Ωであり，筆者の手元にある1981年の資料を見るかぎり，この当時で伝送線配線長は最大で4000フィート（100kbps時），伝送速度は10Mbps（15フィート時）という速度をたたき出しています．

ちなみにこの時代，1979年に登場したMC68000の4MHz版が日本で流通し始めたころなので，当時としてはRS-422は非常に高速な通信手段の一つだったのです．

● 通信分野で多用されるCML

そして，このCMLインターフェースは，いまや通信分野ではなくてはならないインターフェースの一つです．

第2章で解説する光デバイスの送信側/受信側デバイスのインターフェースには，CML方式を採用するものが非常に多く見られます．さらにいえば，光ファイバによるデータ通信のプリアンプの入出力は，総じてCML方式である場合が多いのですが，その理由としては回路構成が異常なほど(?!)に簡単であるからではないでしょうか．

回路構成が簡単ということは，信号の伝達経路が短く，このため高速なスイッチングにも対応することが可能となります．ECLも十分に単純な構成なのですが，CMLは出力トランジスタさえも省いた回路構成なので，さらにこの出力トランジスタ段の分だけ速くなります．

● CMLの特性

CMLは，ターミネーション抵抗などもある程度自由に決められます．ECLの場合には，ターミネーション電圧は$V_{CC}-2V$近辺（回路上では正確には2.09V），そしてトランジスタのエミッタ出力ピン上に電圧振幅を発生させるためにはこのターミネーション電圧に接続されたターミネーション抵抗が必要であり，その抵抗値として50Ωが強く推奨されています．実際，この値でなければECLの回路経路として成り立たないというのが現状です．

それと比較して，CMLでは差動アンプそのものを外部V_{CC}とターミネーション抵抗に頼っているためにECLのような規則はなく，単純に電流経路さえ確保すればあとは次段の差動アンプの動作点の確保だけで動いてしまいます．

抵抗値も100Ωだったり，50Ωだったり，伝送線路の特性インピーダンスに応じて決定でき，自由度がとても高くなっています．

また，CMLの入力電圧範囲は，差動アンプさえ駆動できるのであれば，何Vp-pかなどというのもじつはあまり重要でないスペックの書き方がされています．

Column 4
原理を理解すれば応用も効く

CMLで説明しましたが，CMLインターフェースだからといって，ドライバ/レシーバの双方がCMLインターフェースでなければならないということはありません．最新のFPGAがこれだけいろいろなインターフェースに対応してくると，本来なら異なるインターフェースでありながら，抵抗を付けたり電圧を少し変更するなどの工夫次第で，電気的に問題なく接続できてしまうことがあります．

ディジタル設計者の場合，厳格なDCスペックや振幅電圧について懸念される方が多いように見受けられます．詳細までは追求しなくとも回路の駆動原理をある程度の範囲で知っておけば，諸般の事情で推奨回路を採用できなくても，問題なく動く回路を実現できるものです．臨機応変にデバイスを使いこなそうではありませんか！

表8は，第2章で解説する光モジュールの電気的特性をまとめたものです．表中の入力差動電圧(Input Differential Voltage)の項目に注目してください．最小200mVから最大1900mVまで，広い範囲で受け付けます．あわせて，コモン電圧(Voltage Common Mode Range：動作時の中心電圧)をみても，V_{EE}から850mV以上(このデバイスは，通常V_{EE}はグラウンドレベル/0Vで使う)であれば，V_{CC}を超えても使えてしまいます．

「何という無節操な電気的特性条件なのか！」と思われるかもしれませんが，この光モジュールにかぎらず，ほかの光モジュールも入力側のインターフェースがCMLであればこんなものなのです．

実際，この回路は3.3V駆動で動くLVPECLインターフェースを直結しても正常に動作します．LVPECLの出力電圧範囲/コモン電圧とCMLの入力電圧範囲/コモン電圧が正しく一致します．

では，出力側はどうかというのをみても，この光モジュールのレシーバデバイスの規格を見ると，100Ω終端時には7mA～11mAとなっています．電圧振幅は，最大で800mVというのもわかるでしょう．出力電流をおおよそ8mA程度で計算すると，100Ω終端時には800mVの電圧が発生すると考えられます．

さらに，レシーバデバイスの差動出力電位はCMLの差動アンプに供給するV_{CML}電圧から500mVと規定されていて，コモン電圧を中心に800mVp-pの振幅を取ることになります．

よって，CMLインターフェースをもつこのデバイスの出力を，たとえば3.3V LVPECLインターフェースレベルにもち込むのであれば，V_{CML}に対してLVPECLのコモン電圧点である2.0Vから500mV程度持ち上げた電圧，つまり2.5Vを供給すれば，電圧レベルは“H”レベルのときに2.4V，“L”レベルのときには1.6Vとなり，LVPECLインターフェースの仕様を満たすことができます．

LVDS

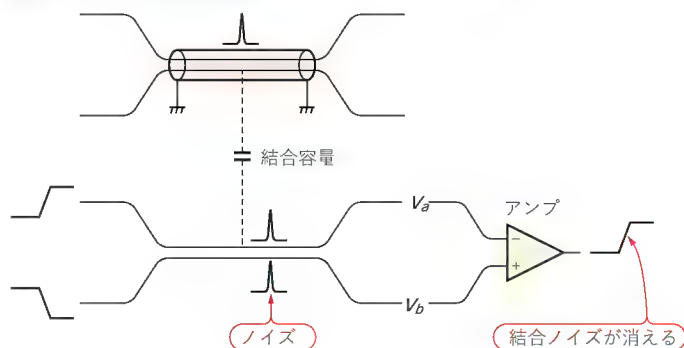
● LVDSとは

LVDSは，PCI Express(第4章参照)やFibre Channelのようなパソコン周辺機器への応用から，最新マイクロプロセッサやFPGAのバスインターフェースの一つとして，いまもっともホットな信号インターフェースの一つです．

LVDSとは，Low-Voltage-Differential-Signalの頭文字をとったもので，直訳すると「低電圧振幅/差動インターフェース」です．

とはいえ，低電圧振幅とだけいってしまうと，何に対して低電圧なのか？という部分が不明なので，このLVDSもSSTLやHSTLと同様に規格が規定されています．

ただし，LVDSはちょっとややこしく，二つの規格団体が規格書を出しているので，若干混乱するのではということを筆者は懸念しています．

〔図14〕LVDSの結合容量

▶ ANSI/TIA/EIA-644(LVDS)規格

LVDSを汎用インターフェースとして利用しようという意図の元に制定された規格です．電気的特性のみが規定されており，コネクタ形状や伝送線路の方式などのどちらかというと機械的な特性は規定されておりません．

▶ IEEE-1596.3

この規格は，もともとLVDSを目的に作られたものではなく，コンピュータシステムのアーキテクチャ規格の中の「SCI：Scalable Coherent Interface」という伝送路規格の一つに，LVDSインターフェースが組み入れられています．

ここではANSI/TIA/EIA-644に基づいた解説を行います．

● LVDSの特徴

差動インターフェースのなかでも，このLVDSは数百Mbpsから数Gbpsの伝送に対応することが可能な，小信号振幅信号インターフェースの一つです．電圧モードでの入出力を行うのではなく，電流モードドライバによる信号レベルの伝送を行い，小振幅電圧を差動インターフェースで出力するために，高速でかつ低ノイズである点が特徴です．

この電圧振幅が小さいということは，スルーレートが支配的になる高クロックでの伝送では重要な要素です．また，それは同時に消費電力を少なくできます．

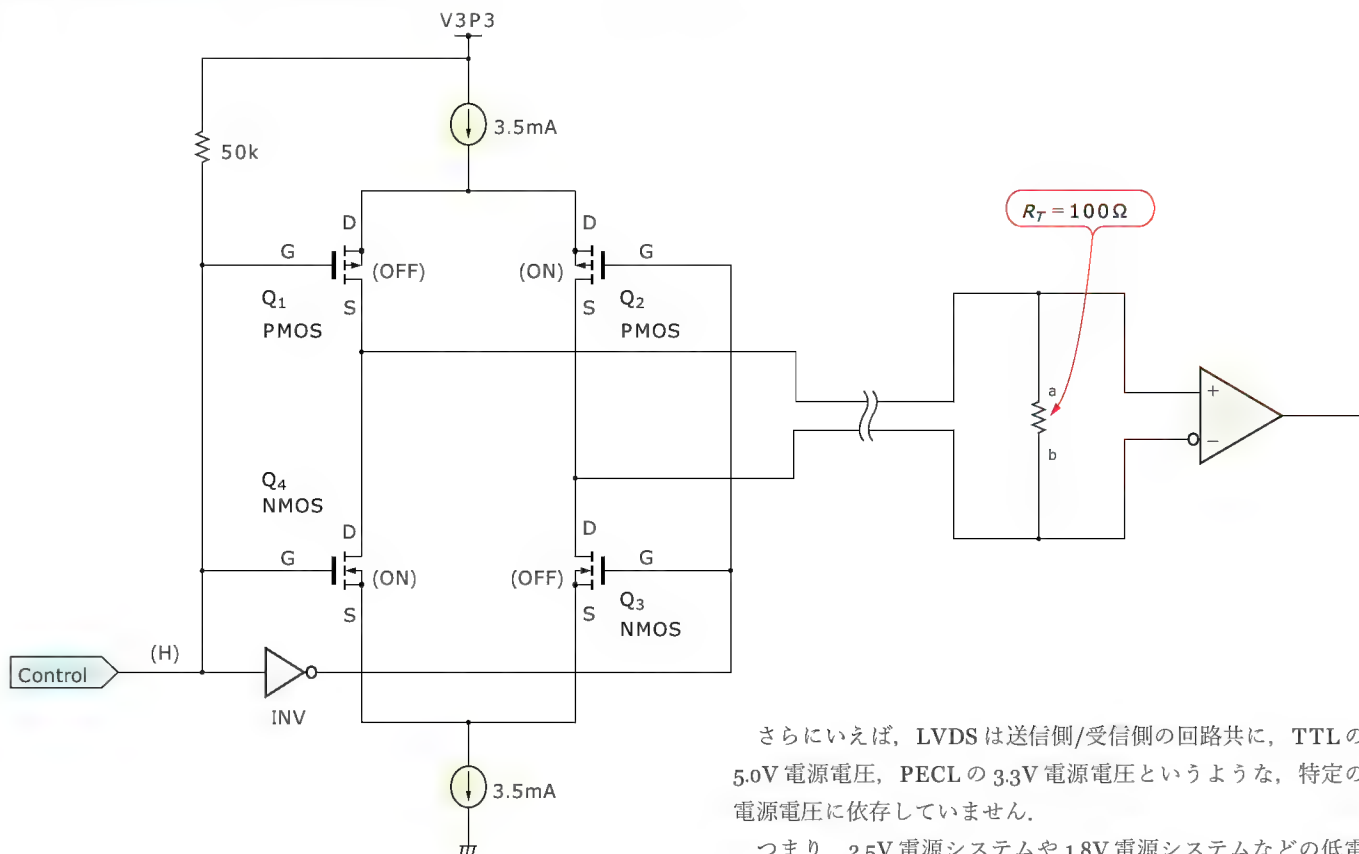
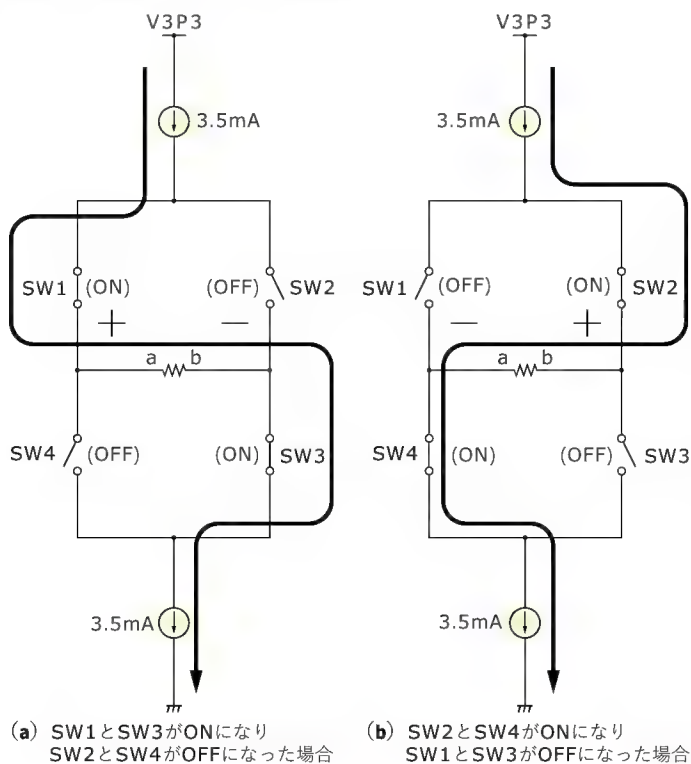
また，基本的に隣り合う，もしくは隣接する1対の平衡(2本で1組の対の信号)ラインで伝送される差動インターフェースは，信号伝送時に発生する磁界が相殺されます．このため，もともと外部へ発生するノイズ量が少ない差動インターフェースですが，発生するノイズは低電圧振幅でさらに小さくなります．

低電圧振幅であると，LVTTLやLVCMOSのようなシングルエンドインターフェースではよく耳にするノイズマージンに対して懸念される方もいるかと思います．

じつは，筆者もシングルエンドではこれに悩まされたのですが，差動インターフェースの場合には，それほど深刻になることもないと考えます．というのは，LVDSは電流の方向で信号レベルが決まるので，仮に電源電圧が多少狂っても平衡ライン上の差動信号には影響はありません．

〔図15〕LVDS出力側ドライバの回路図

〔図16〕LVDS出力側ドライバの動作

(a) SW1とSW3がONになり SW2とSW4がOFFになった場合
(b) SW2とSW4がONになり SW1とSW3がOFFになった場合

さらにいえば，LVDSは送信側/受信側の回路共に，TTLの5.0V電源電圧，PECLの3.3V電源電圧というような，特定の電源電圧に依存していません．

つまり，2.5V電源システムや1.8V電源システムなどの低電圧駆動システムに移行しやすく，かつ特性を損なうことなく信号の伝送ができるという利点があります．

また，自身からのノイズ発生量が少ないということは，隣り合うLVDS信号線対に与える影響，つまりクロストークの発生も小さくなります．もしクロストークが乗ったとしても，このクロストークノイズは差動ペアの両側に同様な強度で乗ることになり，結局このノイズはキャンセルされてしまい，影響はなくなります(**図14**，前頁)．

● LVDS回路の構成と動作解説

LVDSは，MOS-FETで回路を構成することができます．

ECL/PECL/LVPECLやCMLなどの回路で使われているNPN-PNPバイポーラトランジスタと異なり，MOS-FET(モノリシックトランジスタ)で構成できるということは，昨今のCMOSプロセスの進化と相まって製造が楽であり，たいへん大きなメリットとなります．

というのも，すでにご承知のとおりCMOSプロセスはComplex-MOSという名前からもわかるとおり，NMOS-FETとPMOS-FETの組み合わせで作られるものであるため，その製造工程をそのままLVDSの入出力回路に適用できるというわけです．

図15に，NMOS-FETとPMOS-FETで構成したLVDSの出力側ドライバの回路図を示します．入力側レシーバの直前には100Ωの終端抵抗が接続されます．100Ωの終端抵抗の位置を

〔表9〕信号比較表

信号名	LVTTL	LVCMOS	SSTL2-クラス1	LVPECL	LVDS	備考
動作電圧	3.3V	3.3V	2.5V	3.3V	>2.5V	SSTLや差動はドライバの電源電圧とする
動作中心電圧	—	—	1.25V	2.0V	1.25V	
Hレベル電位	2.0V	V_{CC} × 90%	1.65V	2.35V	1.425V	常温・最小値
Lレベル電位	0.8V	V_{CC} × 10%	0.85V	1.60V	1.075V	常温・最大値
電圧振幅	1.2V	3.3V	400mV-800mV	750mV	350mV	SSTL2はおおよそを表す
終端	不必要	不必要	必須	必須	必須	LVTTL/LVCMOSでは一般例を表す
最大伝送速度	166Mbps	133Mbps	333Mbps	800Mbps	2Gbps	筆者の設計目安
用途	一般	一般	DDR/DDR-II メモリ	システム クロック分配	高速伝送 線路部位	筆者設計での主たる用途を示す

MOS-FETスイッチ群の中間にもってきてサンドイッチすると，アルファベットの“H”に見立てられることから，このような回路構成を「Hブリッジ」といいます．

このHブリッジ内の縦に並んだFETをスイッチに置き換え，中心の電圧を検討してみると，もっと回路動作が分かりやすくなります(**図16**)．この回路はSW1とSW3，ならびにSW2とSW4の組み合わせのいずれか片方しかONにならないしくみです．これが大前提となります．

ではまず，SW1とSW3がONになり，SW2とSW4がOFFになった場合を想定してみましょう〔**図16**(**a**)〕．電流はV_{CC}からSW1を経由して，抵抗素子のa端子に流れます．そして，抵抗のb端子→SW3を経由してグラウンドにつながります．つまり，V_{CC}→SW1→抵抗素子→SW3→グラウンドというパスになります．このときの抵抗素子のa端子とb端子を見ると，a端子の電位V_{Ra}はV_{CC}電圧，そしてb端子の電位V_{Rb}はグラウンド電圧となります．

次に，SW2とSW4がONになりSW1とSW3がOFFになった場合を想定してみましょう〔**図16**(**b**)〕．電流の流れは，今度はSW2を経由して抵抗素子のb端子に流れます．そして，抵抗のa端子→SW4を経由してグラウンドにつながります．まとめると，V_{CC}→SW2→抵抗素子→SW4→グラウンドというパスです．抵抗素子の両端の電圧を見てみると，先ほどの例とは違い，a端子の電位V_{Ra}はGND電圧，そしてb端子の電位V_{Rb}はV_{CC}電圧になります．

まとめると，

- SW1/SW3がONのときは，抵抗素子のa端子電位V_{Ra}はV_{CC}
- SW2/SW4がONのときは，抵抗素子のa端子電位V_{Ra}はグラウンド

となり，スイッチの開閉状態に応じて，信号レベルが反転したことになります．

これをもう一度**図15**にあてはめて考えてみましょう．V_{CC}およびグラウンドプレーンとMOS-FETで構成されたHブリッジ回路の間には定電流源があります．V_{CC}側につながるのがソース側定電流源，グラウンド側につながるのはシンク側の定電流源です．

LVDSでは，この定電流源で生成される電流を四つのMOS-FETで構成されたスイッチにより，ソース側の定電流源から電流をターミネーション抵抗に流すことで，ターミネーション抵抗の両端に電圧を発生させ，受信側ドライバに内蔵された差動アンプがその値を読み取ってレベル判定を行います．

ソース側定電流から送出された電流は，ターミネーションを経由して，再度送信側ドライバ内に戻って，シンク側定電流源に流れ込みます．

これらの動きはQ_1とQ_3，そしてQ_2とQ_4のMOS-FETでその電流方向が制御され，ターミネーション抵抗に流れる電流の向きを制御して，“H”レベルか“L”レベルかを決めているということなのです．

ANSI/TIA/EIA-644規格では，定電流源が生成する電流値はおよそ3.5mAです．推奨のターミネーション抵抗値は100Ωとなっているので，この抵抗器の両端に発生する電圧は350mVとなります．

そして，ターミネーション抵抗のある1点をa点とみて，a点からb点の電圧を減算し，その電圧が正であれば“H”レベル，0Vであれば“L”レベルということになります．

なお，差動ペアで構成された配電線路の特性インピーダンスZ_0が100Ωである場合にインピーダンスマッチングすることを想定し，ターミネーション抵抗値の100Ωが決められています．

あわせて，LVDSポートを実装したFPGAやASSPデバイスなども，レシーバとドライバの特性は通常100Ωのターミネーション抵抗に合わせている場合がほとんどですから，基板設計をする際には，特性インピーダンスも仕様どおりにあわせたほうがよいでしょう．

*　　　*

最後に，**表9**に本章で解説したシングルエンドとディファレンシャルインターフェースの比較をまとめておきます．

参考文献

1) 夏谷 実，「FPGAによる高速多ビットのデータ転送　ASICプロトタイプシステム(KM-1)による実現」，EDS Fair 2003 出展者セミナー プレゼンテーション資料，(株)エスケーエレクトロニクス

いくら・まさみ　来栖川電工(有)

第2章 3.35Gbps × 12チャネルで最大40Gbpsの転送にも対応する

パラレル光モジュールによるデバイス間/ボード間通信の現状

小林雄祐

これまで，光ファイバを使った通信は，非常に長距離のデータ通信や，ノイズの多い環境などで使われてきた．ギガビットクラスのEthernetやIEEE1394.bなどにも光ファイバを使ったものがある．その高速化の流れが，近距離であるバックプレーンなどの筐体内や基板間の接続，さらには同一ボード上でのデバイス間データ伝送にも使われようとしている．ここではデバイス間/ボード間の光化の現状と，それを実現する小型の光モジュールについて解説する．（編集部）

1. なぜ光配線なのか

半導体技術の進歩によりLSIに求められるI/Oスピードもますます加速され，従来の多ピンによるパラレル信号から，高速シリアルによるデータ伝送が主流になりつつあります．

ボード間，デバイス間の高速シリアル伝送に使用されるICとして一般的なのは，SerDes（Serializer/Deserializer）と呼ばれるICで，通信ICベンダ各社からさまざまな帯域をサポートする製品がすでにリリースされています．

筆者の会社では，PMC Sierra社のSerDes製品とあわせ，SerDes機能をFPGAの中に盛り込んだALTERA社のStratixGXファミリを高速シリアル伝送市場に紹介しています．実際に，国内機器メーカーからのSerDes製品に対する引き合いは非常に多く，今後もこの傾向は続いていくものと思われます．

また，ADSLに代表されるブロードバンドインターネットの普及により通信機器やストレージ機器の高速化，大容量化が進み，すでに一部の装置においては電気バックプレーンがシステム構築のボトルネックとなってきているといわれています．

〔図1〕電気伝送の限界，光通信路の単距離化

高速ボード設計において，具体的に設計者を悩ませている課題は次のようなものです．

- 電気信号の減衰
- 消費電力の増大
- 伝送線路解析の複雑化
- 基板層数の増大
- 同時動作雑音/EMIの深刻化

伝送速度を低いレートに据え置いてI/O数を増やしても，伝送路の配線数やLSIのピン数が増大し，新たな問題を引き起こします．

CPUやメモリバス，ネットワークの高速化に合わせ，装置内のボード間，チップ間配線にも高速化が求められており，光信号によるデータ伝送を実現するという解決策がすでに見逃せない存在になってきているといわれています．

ギガビット単位の高速信号を，一般的なFR4の基板で1m以上の距離を安定したアイパターンでデータ伝送するのはやさしいことではありません．実際に，複数本のLVDS差動信号をコネクタとケーブルでボード間データ伝送を実現しようにも，適切なコネクタが世の中に存在しないという話をよく耳にします．

このような高速ボード設計における技術者の悩みを解決するため，筆者の会社ではギガビット伝送ボードの設計技術セミナーを昨年より数回開催しています．また，セミナーなどに合わせ，高速電気配線を補完する技術としての多チャネル光モジュールによるパラレル光リンクを紹介してきました．

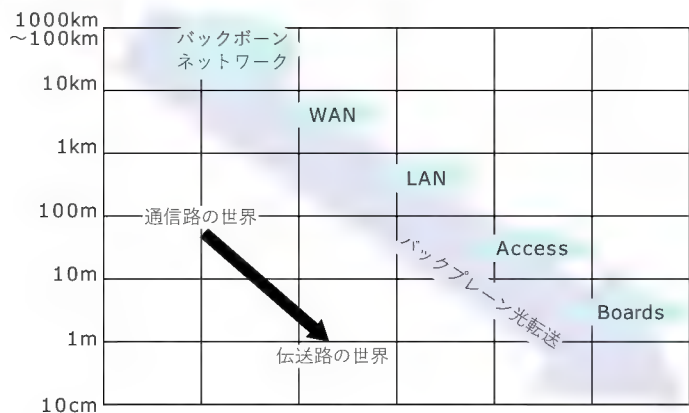
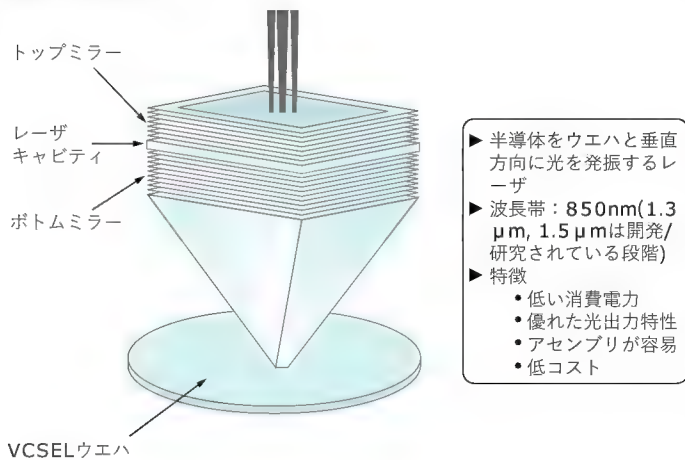
2. 光配線技術の進歩

光の技術もこの数年で飛躍的に伸びており，高速電気伝送の延長線上にある技術の光がより身近に，手頃になってきています（**図1**）．ここではとくに，現在市場が急速に拡大している最

〔図2〕面発光レーザ

トップミラー
レーザキャビティ
ボトムミラー
VCSELウエハ

▶ 半導体をウエハと垂直方向に光を発振するレーザ
▶ 波長帯：850nm(1.3μm, 1.5μmは開発/研究されている段階)
▶ 特徴
• 低い消費電力
• 優れた光出力特性
• アセンブリが容易
• 低コスト

〔図3〕MT型コネクタ

新の半導体レーザと，電気設計技術者からもっとも質問の多い光ファイバ，コネクタに関して簡単に説明します．

● 面発光レーザ(図2)

現在，通信路の世界で一般的に使用されている端面発光レーザ(DFBレーザ/FPレーザ)は，レーザチップの端面(エッジ)から光を出力しますが，レーザ基盤面と垂直方向に出力するタイプの半導体レーザを面発光レーザ(VCSEL：Vertical Cavity Surface Emitting Laser，以下VCSEL)と呼びます．

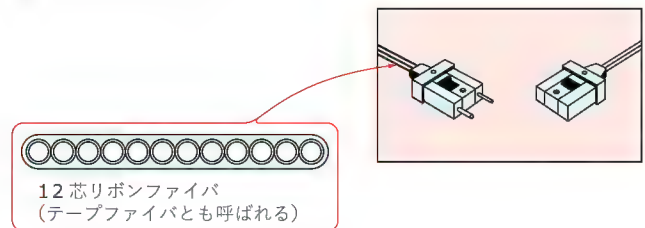
VCSELは光ビームの放射角が狭く，圧倒的な低電流で動作するため，実際にVCSELを使用して高速シリアル電気伝送を置き換える(補完する)ことを目的にしたさまざまな光モジュールが各社よりリリースされています．

VCSELで製品化されているのは，850nmの波長帯域で作動するもののみが製品化されていて，短波長のレーザといえば850nmのVCSELを指しています．数km以上の長距離光伝送が可能な1310nmや1550nmの波長帯域をサポートするものもさまざまな企業や国内外の大学，研究機関でさかんに研究されていますが，未だに製品化には至っていません．

VCSELは，ウエハに結晶させてできるため，テストをウエハのままで行うことが可能で，歩留まりも従来のレーザより格段に向上しています．つまり，通常長距離伝送に使用されている端面発光レーザに比べると，格段に低コストで生産することが可能になっています．同じウエハに並んだ12個のVCSELをそのまま切り取って，12チャネルの光源としてパラレル伝送に使用することもできます．

● 光ファイバ

これは光を通す通信ケーブルで，電気信号を流して通信するメタルケーブルと比べて信号の減衰が圧倒的に少なく，超長距離，高速でのデータ通信が可能となります．また，光ファイバを大量に束ねても相互に干渉しません．

光ファイバケーブルは，用途に応じて大きく二つに分けられます．ガラス製で高速転送に対応するが取り回しが難しいシングルモード光ファイバ(SMF)，プラスチック製もしくはガラス製で転送速度は落ちるが，扱いが簡単なマルチモード光ファイバ(MMF)があります．

▶シングルモード光ファイバ(SMF)

長距離にわたる高速なデータ転送向けで，光ファイバのコアの部分が非常に細いためMMFに比べ折り曲げに弱く，高い加工技術も必要となります．現在の通信装置で標準的に使用されています．

▶マルチモード光ファイバ(MMF)

おもに，LANなどの内部通信に利用されています．SMFに比べ安価で，かつ折り曲げにも強く加工しやすいのが特徴です．伝送距離はSMFに比べて短いのですが，最近は新世代のMMFも製品化されており，伝送距離もどんどん伸びています(VCSELとの組み合わせで500m以上)．接続作業が容易なのも特徴の一つです．

最近発表されているVCSELベースの光モジュールは，MMFに対応したタイプがほとんどで，現在まさにVCSELとMMFの組み合わせにより低価格で光を使用するアプリケーションがますます増えてきています．

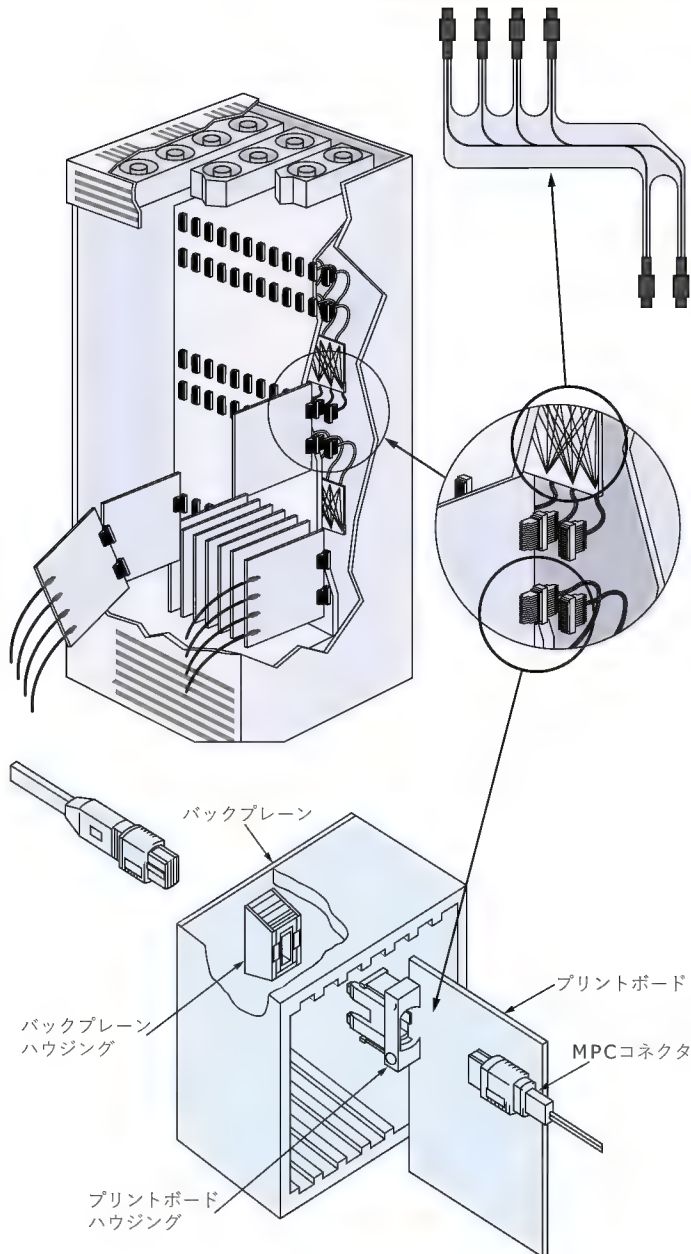
● コネクタ

光ファイバ用コネクタアダプタには，現在多くの種類があり，数多く規格化されています．単芯コネクタのSC型コネクタなどはもっとも一般的なコネクタの一つですが，最近ではさらに高密度/省スペース型のコネクタが登場し，NTTで開発されて世界標準となったMT型コネクタなど，非常に小型で数十芯のファイバを束ねることができるものも登場してきています(**図3**)．

アクセス網の光化などにおける電柱間ケーブルの接続用や光モジュールやコンピュータの光接続に用いられるコネクタとしてますます需要が見込まれているようで，コネクタの世界でも「小型化」がキーワードになり，重要度を増しているようです．

また，最近要望が増えてきたバックプレーンの光化をサポートするコネクタやハウジングも，各社よりリリースされています．大型の通信装置などであれば，このような「光バックプレーンハウジング」を使用することにより，通常の電気バックプレーンと同じように，カードをもったままそのまま挿抜するということもすでに実現可能です(**図4**)．

国内では，主要ファイバ/コネクタベンダとして古河電工，住友電工，フジクラなどが挙げられ，多チャネルに対応した

〔図4〕バックプレーンの光化

ファイバやコネクタなどがすでに製品化されています．その他のファイバベンダとしては，日本MOLEX，住電オプコム，カルテック，北日本電線，日星電気，日立電線，三菱レイヨンなどがあります．

3. 日本における短距離光リンク市場

光技術の進歩も手伝って光が対象にする通信距離はどんどん短くなってきており，同時にすでに紹介したように，高速電気信号での限界も顕著になってきているという話をよく耳にします．

実際に，国内の電気メーカーを訪問した際，ノイズの影響などを回避する方法として光を検討したいという話も少なくはありません[注1]．

長距離伝送を行う通信装置だけではなく，ボード間，チップ間にまで具体的に光を採用する動きが，通信装置以外のアプリケーションで加速しており，従来の電気配線から光配線への置き換えが実際に進んできています．

WDMなどの通信技術で大成功を収めた光技術が，より短い距離で使用されるようになり，すでにコンピュータ，産業機器，放送機器のマーケットにおいては光リンク方式が採用されています．ただ，現在採用されている光モジュールは，ほとんどが1チャネルのトランシーバで，SerDesに接続して高速シリアル信号を複数本単位で伝送するには，このモジュールを複数個並べなければならず，ボードスペース，消費電力，価格についての課題が依然として残ってしまいます．

つまり，1チャネルの光モジュールでは，電気による複数の高速シリアル伝送を完全に補完するまでには至っていないのが現状です．

4. パラレル光モジュールとは

このような状況の中，数年前より注目されているのがVCSELをベースとした多チャネルのパラレル光モジュールによるデータ伝送です．複数の高速I/Oをそのまま光に変換するようなイメージで，大量のデータを安定したアイパターンで伝送することが可能になります．

装置間，ボード間のパラレル光伝送をキーテクノロジとして位置付けた製品が，まさに各社で開発されはじめており，実際に光配線が高速ボード設計者の選択肢の一つになりつつあります．

LANなどに用いられている伝送速度1Gbpsクラスの技術においては，1チャネルの光送受信モジュールと1本のファイバによるシリアルデータ伝送が主流ですが，合計10Gbpsクラスの高速データを接続するには，化合物半導体を使用した高速回路や厳密な温度管理が必要となり，モジュールとしての巨大化，高コスト化と多大な消費電力の増大をもたらしてしまいます．

目安としては，合計5Gbpsを超える装置間などの近距離伝送（1km以内）を低コスト，低消費電力，省スペースで実現するには，光によるパラレルデータ伝送を検討したほうが有利になる

注1：近年の伝送容量の膨大化にともない，装置内の伝送容量も増大の傾向にある．従来の電気信号にて大容量情報を伝送しようとした場合には，並列信号本数を増やすこととなるが，これでは装置の小型化および低コスト化は実現できない．しかし，パラレル光モジュールを採用することで，並列信号本数は格段に低減され，装置の小型化や低コスト化および長距離伝送が実現可能となる〔日立ハイブリッドネットワーク（株）システム部 主任技師 川嶋氏のコメント〕．

場合が多くなります(**図5**).

多チャネルの高速シリアル信号伝送をノイズフリーで実現する“解決策”として，現在注目されているのが「パラレル光モジュール」です．

5. パラレル光モジュールの歴史

パラレル光モジュールの開発は，1990年代前半からLEDなどを光源として研究が始まりましたが，現在各社より製品化されているほとんどすべてのパラレル光モジュールは，VCSELを使用しています．このVCSELこそが，多チャネルの光電変換を圧倒的な小パッケージと低消費電力で実現することを可能にしています．

最初に12チャネルのパラレル光モジュールを製品化したのはInfineon Technologies社で，1998年に発表されました．その後もアレイ状の多チャネルVCSELが安定して供給できることを背景に，2003年4月現在，15社以上のベンダがパラレル光モジュールを供給しており，1パッケージあたり最大で48チャネルまでサポートする製品も発表されています．

コモディティとなったVCSELの発展とともにパラレル光モジュールも発展し，現在の伝送レートは1チャネルあたり1Gbps～3.4Gbps程度までをサポートしている製品がリリースされています．

6. 国内市場における パラレル光モジュールへの要求

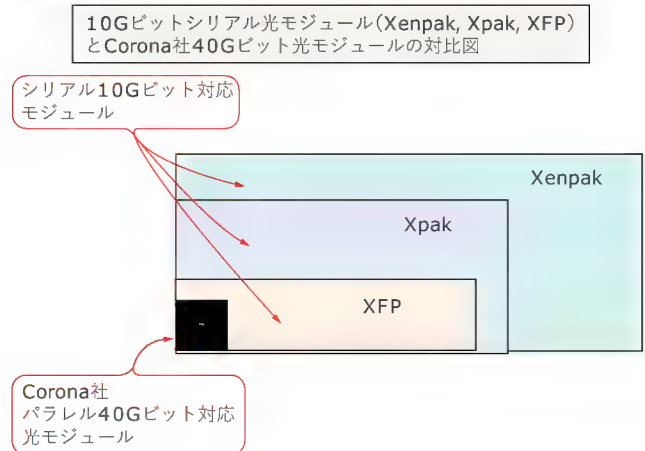
1チャネルの光モジュールにおける接続から，多チャネルのパラレル光モジュールによる光リンク接続へ，市場は確実に推移しています．“複数本の高速電気信号を任意のプロトコルでそのまま長距離転送してしまう”という発想，つまりバックプレーンの光化です．筆者は，パラレル光モジュールを多数の国内メーカーに紹介してきましたが，ターゲットとなるアプリケーションは，当初想像した以上に多岐にわたるものでした．

具体的には産業機器では，

- ICテスタ，LCDテスタ，イメージテスタなど各種テスト装置
- ハイエンド計測器
- 地上波ディジタル放送システム機器/高速画像処理装置
- 業務用ハイエンドプリンタ，ハイエンド印刷機

ネットワーク/コンピュータ機器では，

- ハイエンドスイッチ/ルータ
- ディジタルクロスコネクト
- ADM(Add Drop Mux)
- 携帯電話基地局/無線機
- RNC(Radio Network Controller)
- スーパーコンピュータ
- Sonet/SDH VSRリンク

〔図5〕光モジュールの寸法比較

- 高速SAN(Storage Area Netrwork)リンク，Fibre Channel接続

などに使われています．

高速ボード設計者と話をしてみると，パラレル光モジュールに対する要求は，ほぼ次の数点に絞ることができました．

● 小型であること

もともと電気の置き換えをターゲットとした光モジュールは，とにかく小型であることが要求されます．現在，ICで実現できているボード上に光モジュールが新たに加わることにより，ボードスペースがなくなってしまっては，まったく意味がなくなってしまうからです．これは，光モジュールに接続されるコネクタをどれだけ小さいコネクタで実現するかということと同じで，コネクタが巨大になってしまうとそれだけスペースを取ってしまうことになり，設計が難しくなってしまいます．

● 消費電力が低いこと

携帯電話など，モバイルタイプの製品のみでなく，最近のアプリケーションではネットワーク基幹系の装置においても，消費電力に対する要求はますます強くなってきています．サーバなどに使用されるCPUに関しても(もちろん機種にもよるが)低消費電力タイプが好まれ，消費電力は「低ければ低いほどよい」という声が大半を占めます．今後もこの動きは加速されていくものと思われます．パラレル/シリアル光モジュール問わず，「低消費電力」の光モジュールが今後もシェアを伸ばしていくのではないかと考えています．

● 低コストであること

コストに対する要求は，ますます厳しくなってきています．どの分野においてもこの傾向は顕著になっているのですが，VCSELベースのパラレル光モジュールは合計の伝送レートを考えた場合，1Gbpsあたりの価格をシリアル光モジュールと比較して圧倒的に安くすることが可能です．端面発行レーザを内蔵した高価な光モジュールを使用するのではなく，電気高速I/Oの延長としてパラレル光モジュールを使用するためには，今後もますます価格が下がっていく必要があります．

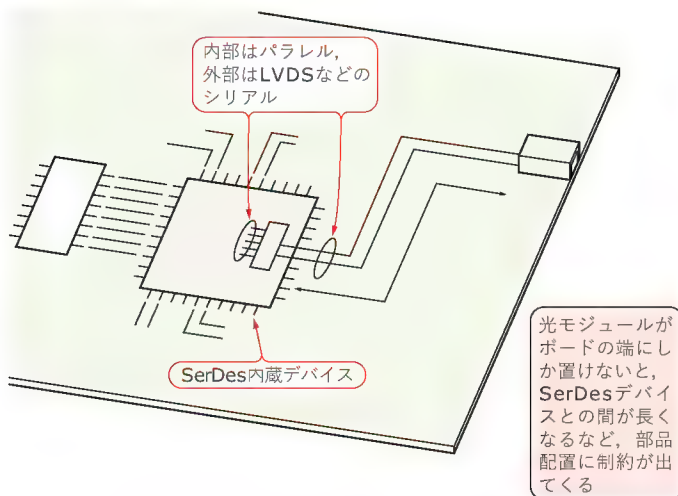
● (光モジュールの紹介とあわせて) 光ファイバ，コネクタに関してアドバイスできること

先ほどもふれましたが，パラレル光モジュールがターゲットにしているアプリケーションは多岐にわたり，光モジュールや光ファイバを使用したことがないユーザーも少なくありません．とくに，装置内のバックプレーンを光化するとなると，バックプレーンに接続されるコネクタの形状や，電気バックプレーンの時と同様にボードを持ったまま挿抜できるようなアーキテクチャがすでに確立されているのかどうかという質問まで，いわゆる“光のハード”部分に関しての知識が必須となってきます．

どのファイバ/コネクタベンダがどのような製品を市場に投入しているか？というような情報も，初めて光モジュールを採用しようと考えているエンジニアには非常に重要な情報で，サポートが必要になってくる分野で，このあたりのハード面での制約によって，システムの構成が変わることもあります．

● ボード上の光モジュールの位置に制約がないこと

この点も意外に思われるかもしれませんが，今後ますます重要になってくるのではないかと思われます．現在でもほとんどの光モジュールは，ボードの端にしか置けないような設計になっています．現在発表されているほとんどのパラレル光モジュールは，バックエンドにSerDesと呼ばれる高速I/Oを内蔵したICの使用を想定しています．SerDesと光モジュールの距離を離してしまうと，1チャネルあたり1Gbps以上の電気信号をFR4の基板上において数本伝送する場合，距離が長ければ長いほどノイズや減衰の影響が深刻になり，ボードの信頼性が低くなってしまうのです(**図6**)．このため，光モジュールの位置にあわせてSerDesなどのICを移動させるのは，その分ボード設計の柔軟性が失われてしまうことにつながります．

また，現在のパラレル光モジュールは一定のエアフローを要求しているのに対し，実際のボード設計では場所によっては風があたりにくい部分が生まれ，それが問題になる場合もあります．光モジュールをボード上のどの部分にでも配置でき，エアフローに合わせてヒートシンクのカスタマイズが可能であれば，そのような悩みをすべて解消することが可能になります．

〔図6〕実装位置の制限

7. 世界最小光モジュール“OptoCube40”

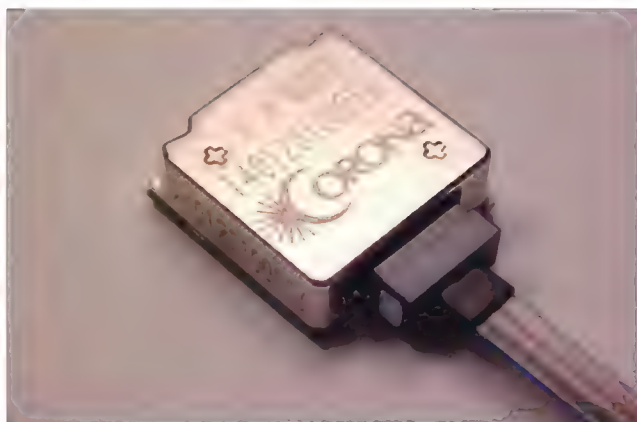
このような市場の要求に対し，合計40Gbps(最大)の光電変換を1チップあたり1W(Typ.)の消費電力と13mm角のパッケージサイズで実現することを可能にしたパラレル光モジュールが，“OptoCube40”(Corona Optical Systems社，以下Corona社)です．1cm^{2}あたりに換算すると約23Gbpsの光電変換を行うことができる計算になります．**写真1**に，OptoCube40の外観を示します．

Corona社は2000年2月に設立され，本社を米国イリノイ州に置くパラレル光モジュール専業メーカーです．OptoCube40は12チャネルのパラレル光モジュールで，2001年よりリリースされています．チャネルあたりの転送レートは100Mbps～3.35 Gbpsをサポートし，12チャネル合計で40Gbpsの光電変換を行うことが可能な光モジュールです．また，業界初の表面実装可能な光モジュールとして13mm角の世界最小パッケージで製品化しています．OptoCube40の仕様を**表1**に示します．

製造工程の自動化を実現したMCM (Multi chip module)タイプのモジュールで，リフローを通すことが可能なことから，通常の半導体部品と同等に取り扱うことも可能です．

光電変換のスピードは，光モジュールに送るLVDSまたはCMLの電気信号のスピードによるため，たとえばXAUI 3.125Gbpsの信号であれば，そのままのスピードで光に変換し，プロトコルもそのまま転送します．そのまま12チャネルを束ねれば合計約38Gbpsになります．100Mbpsから3.35Gbpsであればどのようなスピード，プロトコルでも光にデータを変換し，ノイズの影響を受けることなく安定したアイパターンで最大600mまで高速データを送ることが可能となります．**図7**に3.35Gbpsで通信中のアイパターンを示します．

なお，この光モジュールとStratixを接続し，PCIボード上に搭載した評価ボードを開発中です(**コラム**参照)．

〔写真1〕OptoCube40の外観

〔表1〕OptoCube40の仕様

TX	電気→光
RX	光→電気
ビットレート/チャネル	100Mbps～3.35Gbps
チャネル数	12チャネル
電源電圧	3.3V(±5%)
インターフェース	CML/LVDS
BER	< 10-12(@223-1 PRBS)
サイズ	W13×D13×H4.8 W13×D13×H9.8(ヒートシンク含む)
消費電力	1W(Typ.)
レーザー波長	850nm(VCSEL)

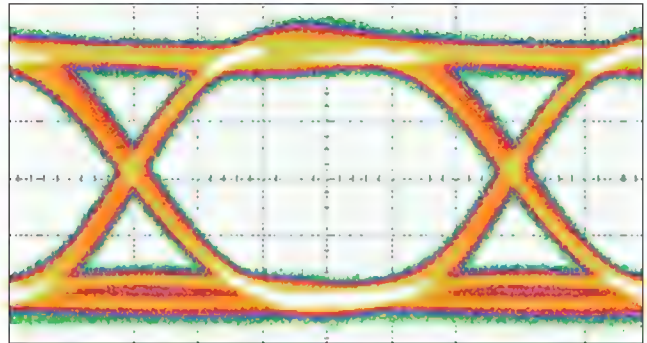
〔図7〕3.35Gbps通信時のアイパターン

8. VCSELベース光モジュールの将来動向

VCSELベースの光モジュールがねらっている市場はショートリーチの市場のみではありません．1チャネルあたり10Gをサポートする VCSELが製品化されれば，10G×nチャネルをサポートとするパラレルモジュールも製品化され，ますます市場を広げていくものと思われます．

Corona社では2004年中に13mm×13mmのパッケージで10Gbps×4チャネルを実現する製品をリリースする予定になっています．

短距離の10Gビット Ethernet接続などは，850nmのVCSELを使用したパラレル光モジュールがどんどん進出してくるのではないでしょうか．また，波長帯域1310nmや1550nmのVCSELで数kmの光伝送を保証する製品も本格的に量産されれば，現在の端面発光レーザ(DFBレーザ/FPレーザ)の市場にVCSELベースの光モジュールが参入してくることになり，世界の大学や企業が，こぞって「本命」であるこの市場をねらっています．

将来的には，ボード上のチップ間インターフェースも光になる可能性があります．たとえば，ソニーが現在開発している次世代プロセッサには，将来的に光インターフェース回路の集積を検討しているようで，CPU同士を高速な光ネットワークでつなぐことを視野に入れているようです．

まとめ

1970年代より，長距離大容量データ伝送技術として発展してきた光伝送の技術は，いま新たな展開を迎えています．冒頭にも触れましたが，光伝送の技術がすでにネットワークの世界を飛び出し，ハードウェアの世界に進出してきました．

多チャネルの光モジュールという製品自体は，すでに安定した品質で量産化されており，この光伝送の技術をサポートし，光バックプレーンを実現するコネクタやファイバも各社より製品化されています．レーザ，ファイバ，パッケージング技術，新しい素材の導入など光技術を取り巻く環境はどんどん進歩し，改善されており，パラレル光モジュールが電気による多チャネル高速シリアル伝送を補完する技術として実際に採用され始めています．まずは装置間やボード間から，新しい技術の光が高速ボード設計者に着実に受け入れられています．

今後もノイズフリーで低消費電力を実現する技術としてパラレル光モジュールが認知され，電子回路設計者にとって光がより身近になるきっかけとなればと考えています．

こばやし・ゆうすけ　(株)アルティマ

Column
光モジュール搭載大規模FPGA評価PCIボード

OptoCube光モジュールを搭載した，大規模FPGA評価用PCIバスボードです(**写真A**)．アルテラ社製FPGA Stratix(EP1S40～EP1S80)を搭載し，光モジュール間は840Mbps-12チャネルで接続されています．その他，SXGAビデオD-Aコンバータ，SO-DIMMやLVDS通信ポートなどを搭載し，WDMやビデオ配信，RAIDシステムなどの設計に最適です．

■問い合わせ先
(株)アルティマ プロダクトマーケティング部　TEL.045-476-2155

〔写真A〕光モジュール搭載大規模FPGA評価PCIボード

第3章　元祖PC/ATからHubInterface，HyperTransportまで

PC/AT互換機チップセットのデータ転送

桑野雅彦

PC/AT互換機は，CPUが486の頃は33MHz，Pentiumが登場して66MHz，Pentium IIIでは133MHzというようにFSB(Front Side Bus)が高速化してきた．さらに最近では，クロックの立ち上がりと立ち下がりの両エッジを使ったDDR動作や，4倍のQDR動作によるデータ転送で，最新Pentium4のFSBは800MHzにも達した．すでに第1章で解説したように，これらは電源電圧や信号振幅の低電圧化により実現できるようになった．また，16ビットから32ビット，64ビットへと広がってきたバス幅も，HubInterfaceやHyperTransportでは逆にビット幅が狭くなっている．ここでは，PC/AT互換機のチップセットのデータ転送について解説する．（編集部）

PCの周辺回路を数個のパッケージにまとめあげた，いわゆるチップセットと呼ばれるLSIは，CPUとともにマザーボードという車の両輪のような存在であるといってよいでしょう．

とくにPentium登場以降は，CPUの性能や機能を引き出すためにはチップセットも「セット」で考える必要があると考えられるようになり，PC向けCPU供給の最大手であるIntel自身がチップセットを積極的に開発，外販するようになりました．現在もなお，チップセットはCPUの性能向上にあわせて要求される性能を引き出せるようにしながら，PCアーキテクチャとの互換性と，新しい周辺機器への対応を図るという，難しい課題を解決しながら発展してきています．

ここでは，PCの歴史とともに歩んできたチップセットの性能向上のカギになったともいえる，チップセット相互の接続部分に注目してみることにしましょう．

● 元祖PC/AT

まず，元祖PCといえるIBMのPC/ATについておさらいしておきましょう．PC/ATは，汎用のICやごく小さなプログラマブルロジックの組み合わせで作られていました．PC/ATのブロック図の概略は，**図1**のようになっています．ISAバスは，PC/ATの前身であるPC/XTの8ビット幅のバスを拡張した，上位互換の16ビットバスです．ビデオカードやFDD/HDDのインターフェースカードなどがISAバスで接続されます．

〔図1〕PC/ATのブロック図

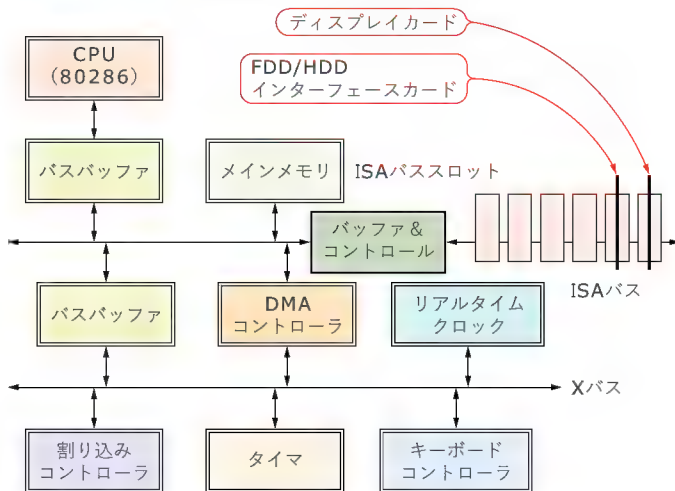
マザーボードの上に標準で搭載される割り込みコントローラ，DMAコントローラなどは，ISAバスからさらにバッファされたバス(Xバスと呼ぶこともある)の上に接続されています．

また，この当時はISAバスとCPUバスの速度が同じだったこともあり，メモリの増設もISAバスに増設メモリボードを入れて行うという方法がとられていました．ISAバスのバスクロックは当初6MHzでしたが，後にCPUの速度向上にあわせて8MHzに引き上げられました．16ビット幅で最短2サイクルで終わるので，ISAバスのバンド幅は8×2(バイト)÷2(サイクル)＝8Mバイト/秒となります．

● 80386時代

PC/ATは，回路図やBIOSのソースコードなどが公開されていたことから，これをデッドコピーした，いわゆるPCクローン機が多数発売されました．そのような中で，Compaqが32ビットCPUである80386を利用し，本家PC/ATを超える性能の互換機を発売するにいたります．

とくに，80386から次の80486への切り替わりの頃には，さまざまなメーカーからチップセットがリリースされる状態になり，まさしく百花繚乱という様相を呈してきます．

この頃のシステム構成は，おおむね**図2**のようになっていました．ピン数の問題から，チップセット部分が分かれているものもありましたが，おおむね図のような構成です．PC/ATのマザーボードに載っていたロジックの大部分がチップセットに吸収されています．CPUが80386になり，データバス幅も32ビット，CPUクロックも16MHz以上となり，ISAバスではバンド幅が足りず性能が出ないため，メインメモリ(DRAM)は

ISA バスから切り離され，CPU バス側のチップセットが制御する形になりました．

シリアルポートやパラレルポート，FDD や HDD インターフェースなどは，PC/AT では拡張ボードで提供されていましたが，パソコンとしてはほぼ必須になっていたことから，これらをまとめあげて ISA バスに直結するだけでよい SuperI/O と呼ばれるチップが登場し，マザーボードは大幅に簡素化されました．図では，キーボードコントローラやリアルタイムクロック（カレンダ時計）は別チップになっていますが，これらを内蔵した SuperI/O もありました．

ISA バスの性能が CPU の性能に対して低すぎることもあって，IBM は PC/AT の後継であった PS/2 で MCA（Micro Channel Architecture）という 32 ビットバスを搭載しますが，仕様を公開しないクローズドなものであったことから，Compaq などの互換機メーカーが ISA の上位互換バスとして EISA バスを提唱して対抗していました．しかし，結局どちらも消えていくことになりました．

● 80486 時代

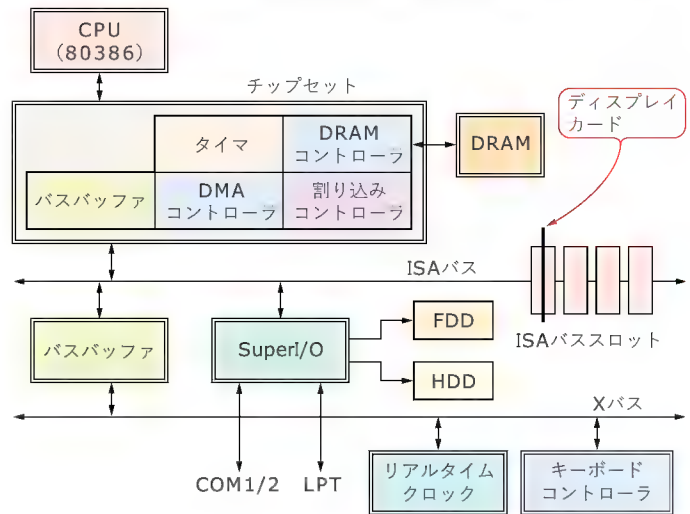
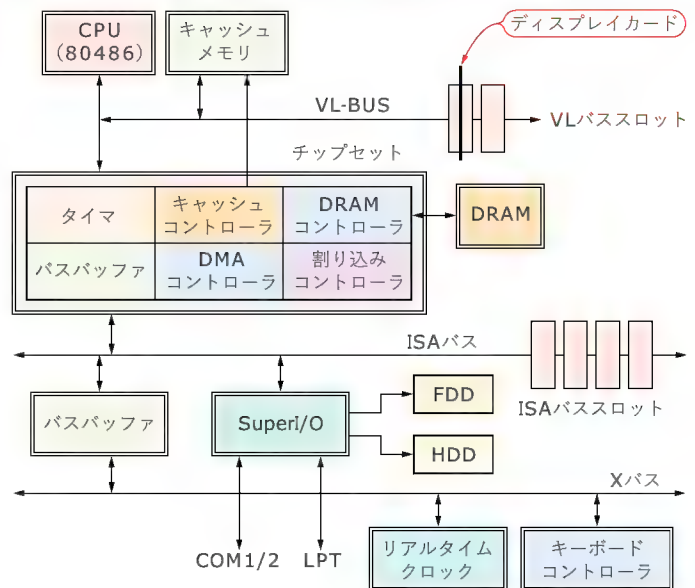
80386 の上位互換である 80486 はパイプライン方式を取り入れ，浮動小数点コプロセッサや，キャッシュメモリなどを内蔵したものでした．ブロック図は**図 3** のようなものです．DRAM の速度が CPU のバス速度に追従できなくなってきたことから，外部に二次キャッシュを用意するのが普通でした．DRAM とキャッシュメモリの制御もチップセットで行います．

この頃になると，PC のアプリケーションもテキストベースからグラフィカルなものが増えてきて，ビデオカードのグラフィック性能などが問われることが多くなってきます．このため，おもにディスプレイカードをターゲットとした高速ローカルバスとして，VL-BUS（VESA-Local Bus ： VESA は Video Electronics Standard Association の略）が設けられました．VL-BUS は，80486 の外部バスをベースに考えられた 32 ビットバスです．最大 66MHz で動作し，266M バイト/秒という速度を実現していました．16 ビットモードがあったり，最大 10 個のデバイスまで接続可能であるなどという特徴がありましたが，アドレスとデータをマルチプレクスしない 32 ビットバスであることから，信号線の本数も非常に多いうえ，80486 という CPU のバス仕様に依存したものであったこと，Intel がオンボードのチップ間接続のためのバスとして PCI バスを提唱し，普及につとめたことなどがあいまって，80486 時代の終焉とともに消えていきました．

この頃までは，I/O もその多くは ISA バス接続で間に合わせている例がよく見られました．

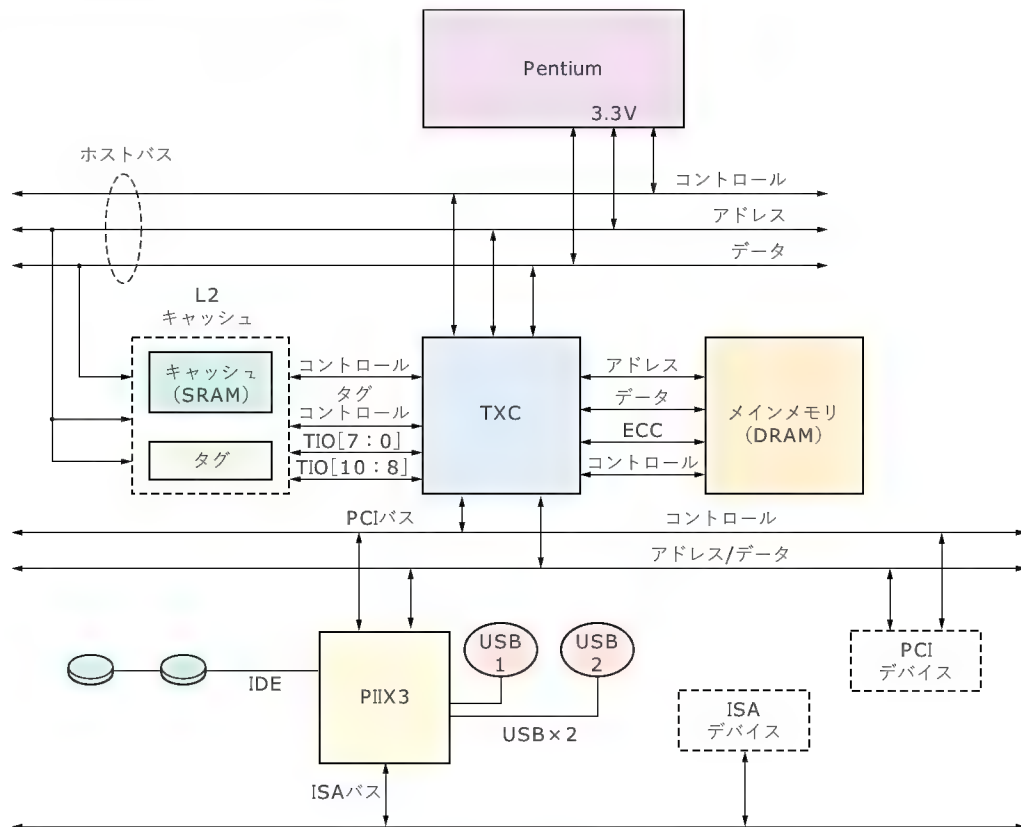
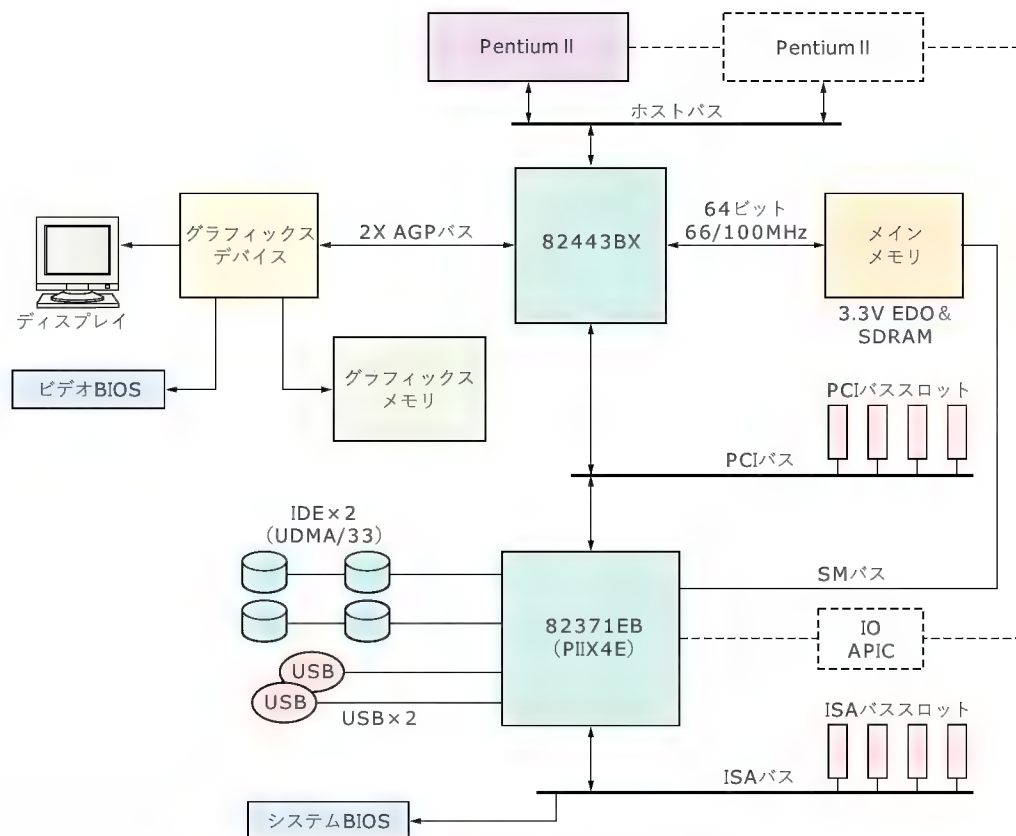
● 430HX（Pentium 時代）

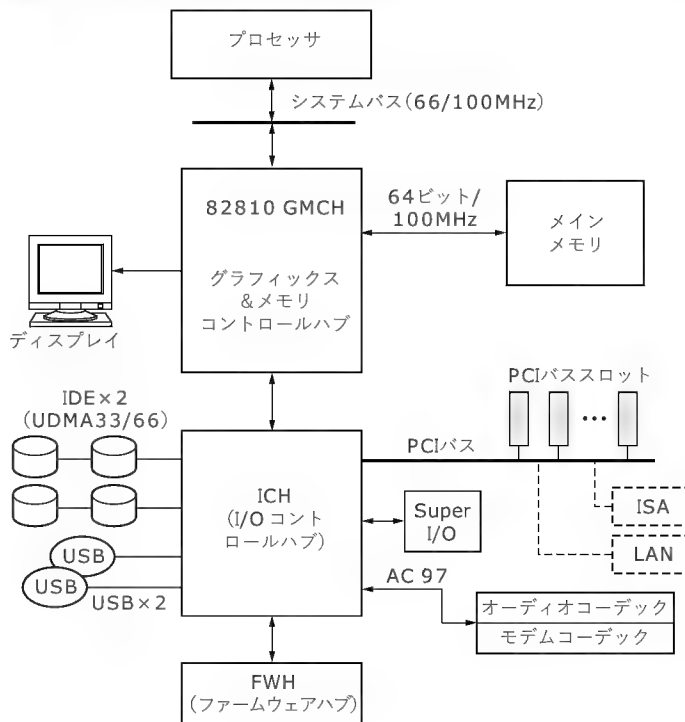
80486 の次に登場した Pentium の時代になると，CPU の性能向上とともに，外部バスの高速化への要求は一段と厳しくなり，Windows の普及とともにグラフィックだけでなく，HDD アクセスも高速化が必要となってきます．Intel は PCI バスの本格的な普及に本腰を上げてきます．

〔図 2〕80386 時代の代表的な PC/AT 互換機のブロック図

〔図 3〕80486 時代の代表的な PC/AT 互換機のブロック図

Intel は，当初はあまりチップセットの開発に熱心ではなく，Zymos 社などから OEM 供給してもらうなどといった状態でしたが，PC への新しい技術導入と性能向上にはチップセットの改良も不可欠と考えたためか，開発に本腰を入れはじめます．

Pentium 時代の代表的なチップセットである 430HX を使ったシステムのブロック図が**図 4** のようなものです．33MHz 動作で最大 133M バイト/秒の伝送速度があり，プラグアンドプレイにも対応した PCI バスがメインのシステムバスとして一般的なものとなりました．

430HX チップセットでは，図のように CPU バスと PCI バスの間に TXC，PCI バスと ISA バスの間に PIIX3 がバスブリッジとして配置されます．地図で上が北になることになぞらえて，TXC をノースブリッジ，PIIX3 のほうをサウスブリッジという

〔図4〕430HXチップセットのシステム構成

〔図5〕440BXチップセットのシステム構成

〔図6〕810 チップセットのシステム構成

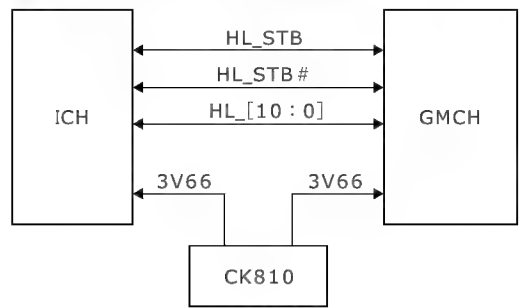
〔図7〕ハブインターフェースの信号

呼び方をすることが一般的なならわしとなっています.

信号レベルは，すべて3.3V系のTTLとなっています.

TXCはCPUバスとPCIバスのブリッジのほか，DRAMとキャッシュメモリ制御を行うシステムコントローラとなっており，PIIX3のほうはPCI/ISAのブリッジに加えてIDEやUSBといった負荷の高いI/O関係の制御を担当します．ビデオカードやSCSIアダプタ，ネットワークカードなど，データ転送速度を要求するものなどもPCIバス上に置くことが一般的となり，拡張バスの主役となりました.

● 440BX(Pentium II時代)

Pentium II時代の代表的なチップセット440BXを使ったシステムのブロック図は，**図5**のようなものです．ノースブリッジが82443BX，サウスブリッジが82371EB(PIIX4E)という組み合わせです．Pentium IIでは，二次キャッシュまで内部に取り込まれたので外部キャッシュがなくなりましたが，基本的な機能分担や，相互にPCIバスで接続されるところなどは430HXと同じです．目新しいところとしては，AGPバスが設けられていることがあげられます.

Pentium時代の後期からPentium IIの時代になると，PCでの3D表示や，動画再生などもあたりまえのように行われるようになり，それとともにディスプレイカード用のデータトラフィックも格段に増加していきました．PCIバス接続では，貴重な共通バスのバンド幅の多くをグラフィックカードに食われてしまうことから，グラフィック関係専用のバスとしてPCIバスとは独立したAGPバスが追加されたというわけです.

AGPバスは，32ビット幅のPCIバスを66MHz動作にしたようなもので，最大転送速度は266Mバイト/秒でしたが，440BXに搭載されたAGPX2バスは，66MHzの基準クロックとSTB/STB#の位相をずらしたストローブ信号を使ってクロックの両エッジアクセスにすることで2倍の533Mバイト/秒のバスバンド幅を確保しています.

また，Pentium IIからはホストバスがTTLではなく電圧レベル1.5VのAGTL+に，さらにPentium IIIは1.25VのAGTLとなっています．AGPやSDRAMは3.3Vのままです.

● 810(Celeron/Socket370時代)

その後のPCの普及と高性能化は進み，ビデオだけでなく，ハードディスクなどのストレージ系デバイスも440BX時代のUDMA/33(33Mバイト/秒)からUDMA/66(66Mバイト/秒)と高速化さていきます．サウスブリッジがPCIバス接続ではディスクアクセスがPCIバスのバンド幅を食いつぶすということになってしまうことや，ISAバスの原則廃止など受けて，チップセットの構成が大幅に変更されることになりました．型番も新たに800番台シリーズが採用されます.

810チップセットを使用したPCの構成は，**図6**のようになっています．ノースブリッジ，サウスブリッジと呼ばれていたチップセットは，それぞれメモリコントロールハブ(82810)，I/Oコントロールハブ(82801A)という名前に変更され，ハブの間は専用のローカルバス(HubInterface)で接続されます．ハブインターフェース部分をHubLinkやHubLink Architectureと呼ぶこともあります.

810のハブインターフェースは，66MHzのクロックで動作する電圧レベル1.8Vの8ビットバスです．二つのストローブ信号を使い，1クロックあたり4バイトの転送を行います．最大転送速度は266Mバイト/秒と，PCIバスの2倍の性能をもっています．これによって，ディスクとメモリ間のデータ転送においてPCIバスがボトルネックになったり，大量のデータ転送にともないPCIバスが占有されてしまうことを防いでいます.

ハブインターフェースの信号接続関係は，**図7**のようにきわめてシンプルなものです．3V66は66MHzのクロック，HL[10:0]が11本のコマンド/データラインです.

また，HL_STBとHL_STB#がそれぞれ位相のずれたストロ

ーブ信号です．周期はクロック周波数と同一で，それぞれの立ち上がり/立ち下がりのエッジでデータ転送を行うことで，1クロックあたり4バイト分のデータ転送を実現しています．

● 845G（Pentium4時代）

ハブインターフェースによるチップセット間接続という考え方は，その後も継承されていきます．Pentium4用のチップセットである845Gも同様に，メモリコントローラハブとI/Oコントローラハブに分かれています．845Gチップセットを使ったPCのブロックは**図8**のようになっています．ハブ相互の間の接続もやはりハブインターフェースですが，電圧レベルが810などのハインターフェース（HL1.0）の1.8Vから1.5Vに引き下げたモード（HL1.5）が用意されました．HL1.5も伝送速度はHL1.0と同じ266Mバイト/秒です．

● E7500（Xeonサーバ用）

一般のPC用とは別に，Xeonを使用したサーバマシンをターゲットにしたのがE7500チップセットです．E7500のチップセット間接続例を**図9**に示します．サーバの場合，大量のインターフェースカードが使用されることもあり，複数のPCIバスを実装することができるようになっています．

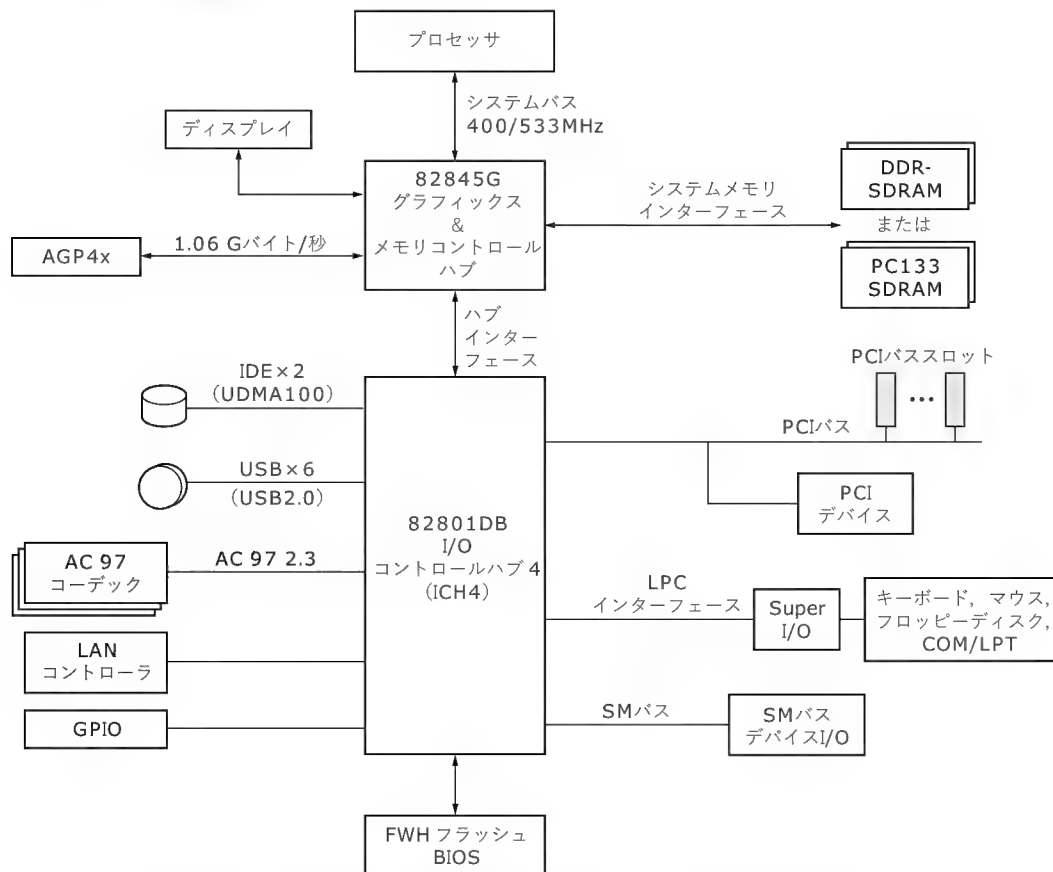
MCHとI/Oコントロールハブ（ICH3-S）だけを切り出してみると，この部分の構成は845Gと同じ，HI1.5による接続になっています．HI1.5ではダウンストリーム（CPUからI/Oコントロールハブへの方向）のアドレス空間は，PCIバス自身が32ビット空間であることから32ビット分ですが，逆のアップストリーム方向では64ビットのアドレス空間をもっており，4Gバイトを超えるメモリ空間への転送が可能となっています．

E7500で特徴的なのは，MCHからPCI/PCI-Xに直接ブリッジするP64H2を最大3個までつなげられるようにしている部分です．P64H2は33MHzまたは66MHzのPCIバスのほか，66MHz，100MHz，133MHzのPCI-Xとインターフェースするチップです．

PCI-Xは，PCIと比べて飛躍的にデータ転送速度が上がっていることから，P64H2用のHubLink側も改良され，HL2.0となっています．HL2.0は66MHzのクロックで動作する点はHL1.5と同じですが，データ幅が16ビットになり，さらに1クロックで8回の転送を行うことで，最大1Gバイト/秒の転送速度を実現しています．

HL2.0の伝送信号は，

- HL［21：20］：ECC信号
- HL［18：0］：ハブ間接続信号
- PSTRBF　：下位8ビット第一ストローブ信号
- PSTRBS　：下位8ビット第二ストローブ信号

〔**図8**〕**845Gチップセットのシステム構成**

- PUSTRBF：上位8ビット第一ストローブ信号
- PUSTRBS：上位8ビット第二ストローブ信号
- CLK66　：66MHz基準クロック

となっています．

- E8870チップセット

E8870チップセットは，大規模サーバを意識したものです．図10のように，最大4個のプロセッサに一つのSNC（Scalable Node Controller）が接続され，SNCの下にはさらに二つのSP（Scalability Ports）が引き出されています．各SPは最大3.2Gバイト/秒の伝送速度をもち，I/Oハブと接続されます．PCIバスや旧来のI/O関係は，I/Oハブから引き出されます．

〔図9〕E7500チップセットのシステム構成

さらに，SNCから引き出されるSPとI/Oハブの間にポートスイッチ（LANのスイッチングハブのようなもの）を接続することで，CPU＋SNCのペアを複数接続したシステムも構成可能となります．

〔図10〕E8870チップセットのシステム構成

- V-LINK（VIA：KM266/KN400）

Intel以外のチップセットメーカーの製品も同じような変遷を遂げています．PCIバス接続によるノースブリッジ/サウスブリッジという構成のあと，やはりハブインターフェースとよく似たチップセット間専用バスを設けるようになりました．VIAのKM266チップセットを使用したシステムブロックを図11に示します．Intelのハブインターフェースに相当する部分はV-LINKという名称のバスになっています．66MHzの8ビットバスで266Mバイト/秒を実現しているということで，Intelのハブインターフェースと同様に，1クロックで4バイト分の転送を行っていることが読み取れます．

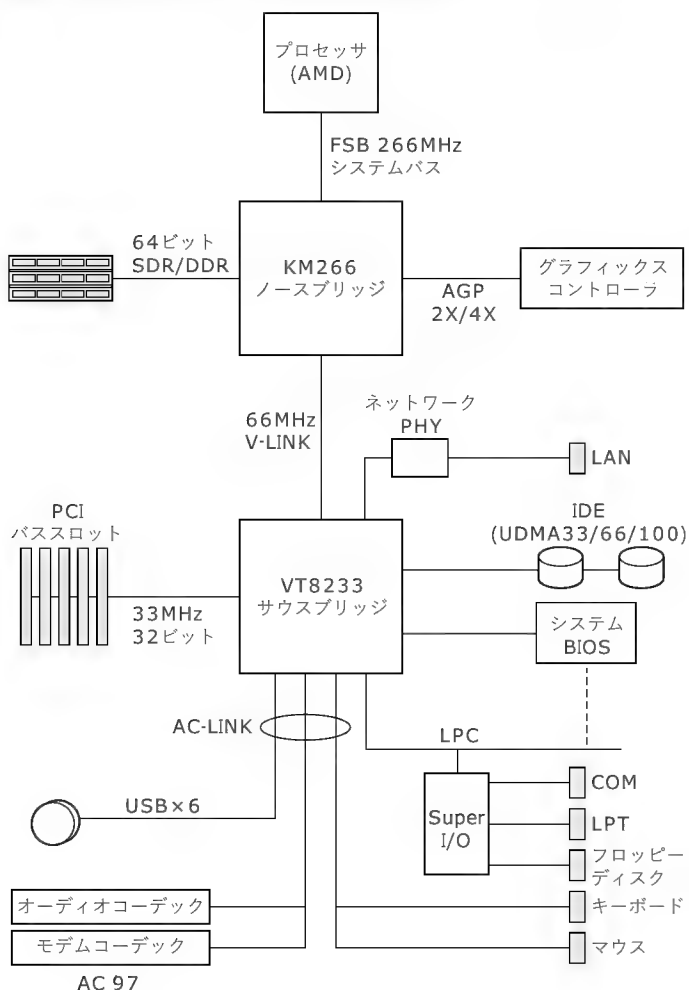
上位のKN400チップセットでは，V-LINKが8X V-LINKと名を変え転送速度もV-LINKの2倍の533Mバイト/秒に引き上げられています．

- MuTIOL（SIS）

SISのチップセットも，チップ間を専用バスで接続する方式をとっています．SISの専用バスはMuTIOLという名称がつけられています．伝送速度はSISの645チップセット（SIS645とSIS961の組み合わせ）に採用されたものはIntelのHL2.0と同様にバスクロック66MHzでデータ幅は16ビットあり，1クロックで4回の転送を行うことで533Mバイト/秒，さらに新しい648チップセットでは，これをさらに引き上げたMuTIOL1Gとなり，1Gバイト/秒のバンド幅をもつバスになっています．

MuTIOL採用のチップセットの構成は，図12のようになります．ノースブリッジ側でメモリやビデオカード関係を，サウスブリッジ側でI/O関係の面倒を見るという考え方は他社のものと同じですが，内部の考え方が変わっています．SISのチップが特徴的なのは，MuTIOLで接続されたチップセット間で複数のトランザクションをキューイングし，処理を並列実行できるようなしかけを設けて，性能向上を図っている点です．これをHyper Streamingと呼んでいます．

図13は，サウスブリッジ側の内部構成です．一つのMuTIOLポートに対するアップストリーム方向，ダウンストリーム方向のそれぞれについて複数の伝送チャネルをもっており，メッセージ（パケット）ベースでの伝送制御を行います．通常であれば，下位の低速デバイスへのアクセスを行うと，動作が完了するまでバスが占有されてしまいますが，これを下のようにリクエスト/レスポンスというパケットに分割して，リクエストパケットを発行後MuTIOLバスを開放し，相手からの応答

〔図11〕KM266チップセットのシステム構成

を待ってレスポンスパケットを受け取るという方法にすることで，バスの有効利用を図るというわけです．この方法をスプリットトランザクションと呼んでいます．

● HyperTransport

AMDが発表したHyperTransport I/O Linkはチップセット間にとどまらず，半導体チップ同士を1対1で接続するための汎用高速I/Oバスとして提唱されたものです．AMDは各社に呼びかけ，HyperTransportテクノロジコンソーシアム(`http://www.hypertransport.org/`)を結成しました．当初8社の集まりであったコンソーシアムも，4月1日現在で49社が参加するものになっています．

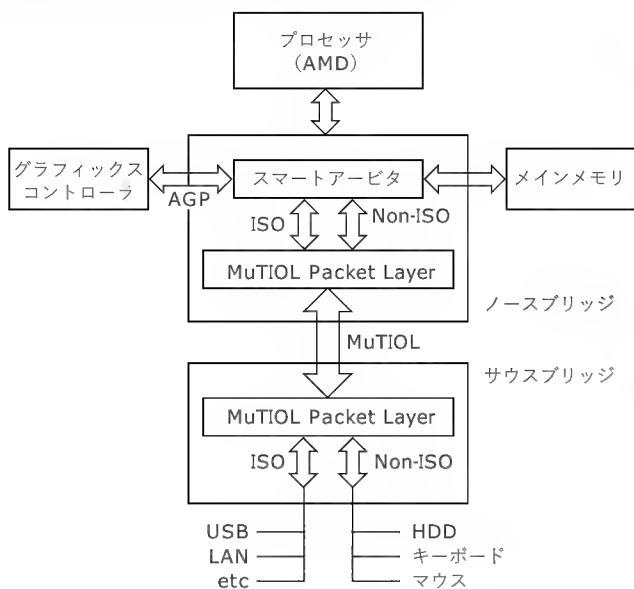
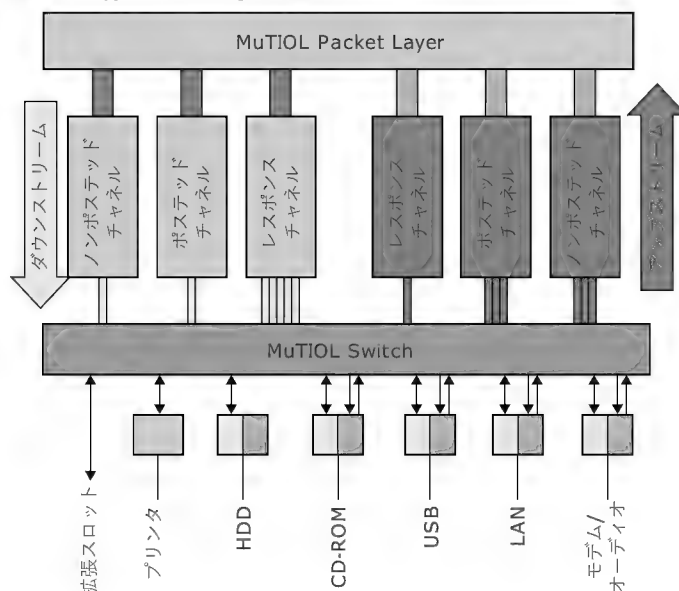
HyperTransportの大きな特徴は，

(1) ポイントツーポイント接続であること
(2) 最大12.8Gバイトのアドレス空間をもつ
(3) データバス幅が2/4/8/16/32ビットから選択可能
(4) 800Mbps(バス幅2ビット，200MHz動作)から最大51.2Gbp(バス幅32ビット，800MHz動作)まで用途に応じて選択可能

となるでしょう．

HyperTransportは，データやコマンドなどをパケット単位でやりとりする方式をとっており，信号線は**図14**のようなごくシンプルなものです．

CAD(Comamnd/Address/Data)には，コマンドやアクセスするアドレス情報，データなどの情報が乗ります．HyperTransportではCAD信号の本数(ビット数)は，2ビットから32ビットまでの5種類が定義されており，要求される伝送速度やチップのピン数など，用途や目的に応じて選ぶことができるようになっています．図でもわかるとおり，HyperTransportは単方向の伝送線路を上りと下りそれぞれ独立でもたせることで，全二重

〔図12〕MuTIOLチップセットのシステム構成

〔図13〕Hyper Streamingの構成

での通信を行えるようにしています.

CTL(Control)は，現在CAD上にあるのがコントロールパケットなのか，データパケットなのかを示す信号です.

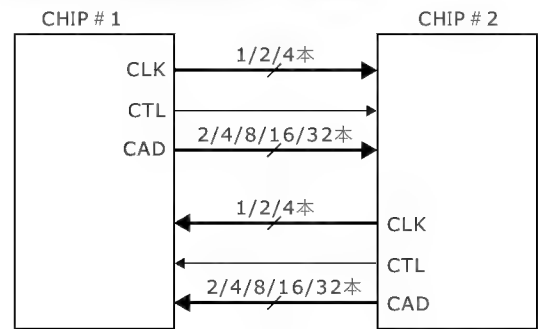
CLK(Clock)は，CADやCTL信号のための同期クロック信号です．CLKはCAD信号8本ごとに1本用意されます．つまり，CADが32ビットならクロックは4本，CADが16ビットならクロックは2本になります．CTLはCAD[0]，つまりCADの最下位ビットと同じクロックが使われます.

このように8ビットごとにクロックを分割することで，基板上での配線長さの違いなどによるスキューの問題を回避しています．クロック周波数は，現在200MHzから800MHzまでの5種類が定義されています.

HyperTransportのCADビット数とクロック周波数による伝送速度の一覧を**表1**に示します.

HyperTransportの信号はポイントツーポイントで接続されることになっていますが，LSIデバイス内部でHyperTransport信号を中継して，次のHyperTransportデバイスとつなぐという方式で複数のデバイスをチェーン，あるいはツリー状に接続することもできるようにしています．**図15**は，この一例を示したものです．**図15**で，Pとあるのはプライマリインターフェースブロック，Sはセカンダリインターフェースブロックを示します．円筒形のものはトンネルと呼ばれるもので，HyperTransportパケットを細工せずにそのままもう一方のポートに流すものです.

HyperTransportは，このようにブリッジとトンネルを組み合わせることで，自由度の高いチップ間接続を実現しているというわけです.

〔図14〕HyperTransportによる接続

HyperTransportによって，チップ間接続方式の標準化が図られることで，さまざまな機能ブロックをもったLSIを自由に結線して利用できるようになることが期待されるといえるでしょう.

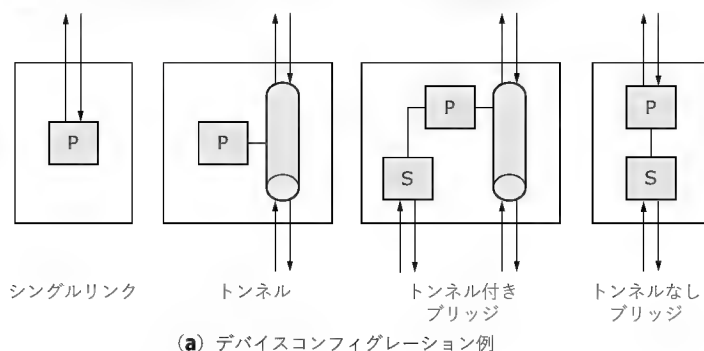
*　　　　*

PCのCPUは，まだまだ高速化の一途をたどるでしょう．さらに次世代になれば，チップセット間のデータ転送量も増大するでしょう．ISA接続，PCI接続，そして専用接続バスへと進んできたチップセット間接続も，これに呼応してさらなる高速化が図られていくのでしょう．また複数の伝送チャネルの使い分けがなされていくような方向に向かうのかもしれません.

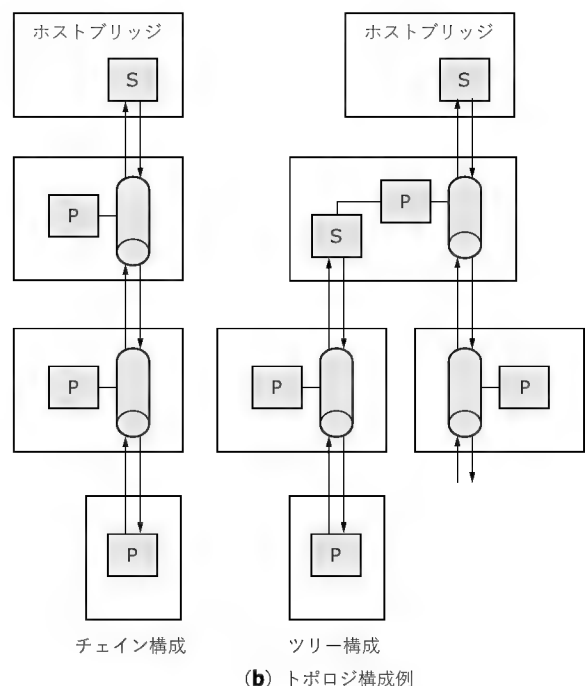
くわの・まさひこ　パステルマジック

〔図15〕HyperTransportによるチップセットの接続構成例

(**a**) デバイスコンフィグレーション例

(**b**) トポロジ構成例

〔表1〕HyperTransportの伝送速度

単位：Gbps

		CADバス幅				
		2ビット	4ビット	8ビット	16ビット	32ビット
CLK周波数	200MHz	0.8	1.6	3.2	6.4	12.8
	400MHz	1.6	3.2	6.4	12.8	25.6
	500MHz	2.0	4.0	8.0	16.0	32.0
	600MHz	2.4	4.8	9.6	19.2	38.4
	800MHz	3.2	6.4	12.8	25.6	51.2

PCI Express規格の概要

里見尚志

PCI ExpressはPCIの後継規格として，PC/AT互換機だけでなく，今後のコンピュータシステム全般で通用する標準拡張バスとして規格化されたバスである．低電圧差動信号伝送，ポイントツーポイントで送受信独立の通信チャネル，パケット化されたスプリットトランザクション，リンク構成の違いによる高いスケーラビリティなど，ISAやPCIとはかなり異なるバスとなる．(編集部)

はじめに

PCI Expressは，今後10年以上を見越した，従来のPCIバスに代わるI/Oバス規格である．2002年夏に最初の規格が策定され，対応製品の開発もすでに始まっている．本稿ではPCI Expressについて，その概要とアーキテクチャについて解説する．

PCI Expressの規格については日々アップデートされており，最新の情報については後述するPCI-SIGやIntelのPCI Express関連情報を参照していただきたい．また，本解説では現在ドラフトとしてレビューされている情報も含まれており，正式な規格では変更となる可能性があることを留意してほしい．

● PCI Expressが登場した背景

現在および今後のコンピューティング/コミュニケーションプラットホームのインターコネクトで要求される性能は，次のような技術革新により，PCIなどの既存パラレルバスの能力を越えつつある．

- 3GHzを越えるような高速CPU
- メモリの高速化
- 高性能グラフィックスコントローラ
- Gigabit Ethernetや10Gビット Ethernetといった高速ネットワーク
- 高速なストレージ/通信インターフェース
- 接続する周辺機器の高速化

これにより現在のPCIバスを大きく越えるバンド幅や柔軟性をもつインターコネクト(I/Oバス)が要求されている．

PCI Expressはこのニーズに応えるためのI/Oバスとして登場し，次のような要求を満足するものとなっている．

(1) 幅広いマーケットセグメントと新たなアプリケーションのサポート

デスクトップ，モバイル，ワークステーション，サーバ，コミュニケーションプラットホームや組み込み機器といったものをサポートする単一のI/Oアーキテクチャ．

(2) 低コストで大量のソリューションの提供

システムレベルで現在のPCIと同等かそれ以下のコスト．

(3) 多様なプラットホームインターコネクト形態

チップ間接続，コネクタやケーブルによるボード間接続など．

(4) 新たな機構的フォームファクタ

モバイル，PCIと同形状，モジュール，カートリッジ．

(5) PCI互換のソフトウェアモデル

PCIで使用されているコンフィグレーションメカニズムをそのまま使用，既存OSやデバイスドライバを変更せずに使用できる．PCI Expressで新たに追加された機能のコンフィグレーションは，既存のコンフィグレーションを模範とする．

(6) 高性能

オーバヘッドや遅延を抑えることにより，アプリケーションレベルでのデータバンド幅やリンク使用効率を増大させる．またピンあたりのバンド幅を大きくし，全体でのピン数を削減する．レーン数や伝送周波数によりスケーラブルな性能を実現．

(7) 進化した機能

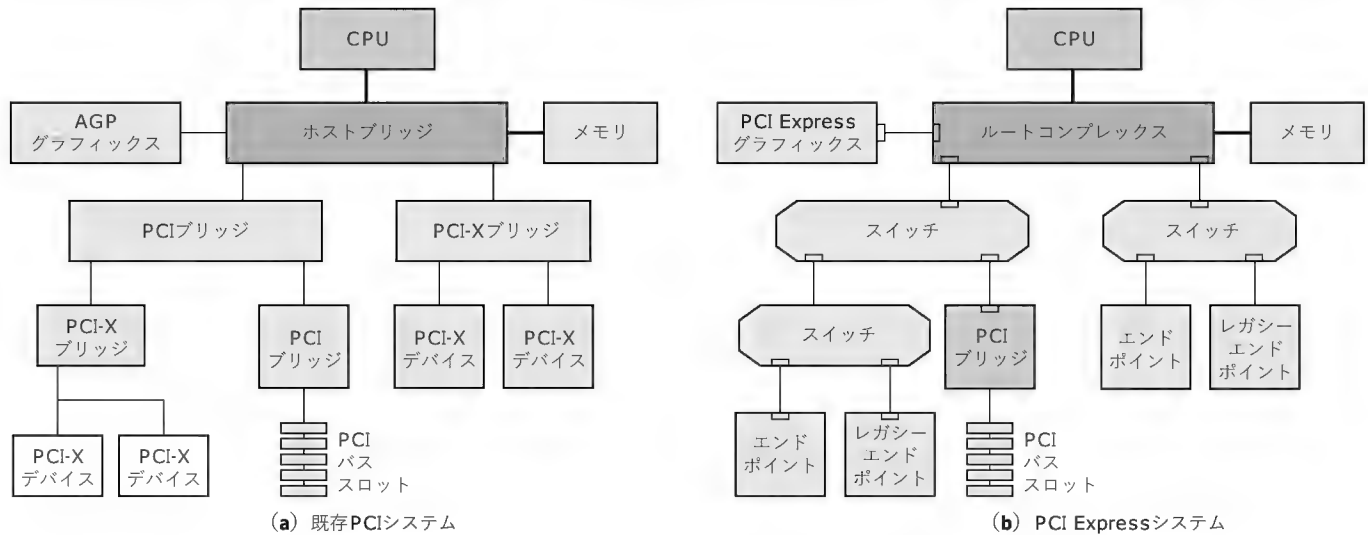
異なるデータタイプや伝送順序ルールの包含，異なるQoS(Qualities of Service)の提供によるサービスの差別化，電源管理，既存PCIおよびサイドバンド信号を使用しないPCI Express固有のホットプラグ/ホットスワップ，リンクレベルおよびエンドツーエンドでのデータ完全性，エラーハンドリングとロギング，半導体プロセス技術への非依存性(送信側/受信側で異なるDCコモンモード電圧)，テストの容易性(電気的適合性試験を試験装置との簡単な接続で実施可能)など．

● PCI Expressの歴史

PCI Expressは当初3GIOやSerial PCIと呼ばれていた「第3世代I/Oテクノロジ」[注1]である．2001年3月にIntelが3GIO，同年7月にPCI-SIGがSerial PCIと呼ばれる構想を発表し，2001年8月になってPCI-SIG，IntelとともにCompaq(現

注1：第1世代はISA，第2世代はPCI/PCI-Xを指す．

〔図1〕既存PCIシステムとPCI Expressシステムの例

(a) 既存PCIシステム　(b) PCI Expressシステム

Hewlett-Packard), Dell, IBM, Microsoftが参加してコード名「Arapahoe」としてワーキンググループが発足，新たなシリアルI/Oインターコネクトアーキテクチャの策定が開始された．規格の名称としては3GIOと呼ばれていたが，2002年4月になってPCI Expressという正式名称[注2]となった．最初の規格となるRevision 1.0のドラフト版リリース/ファイナルレビューとともにPCI-SIGに移管され，2002年7月22日に「PCI Express Base Specification Revision 1.0」，「PCI Express Card Electromechanical Specification Revision 1.0」がリリースされた．

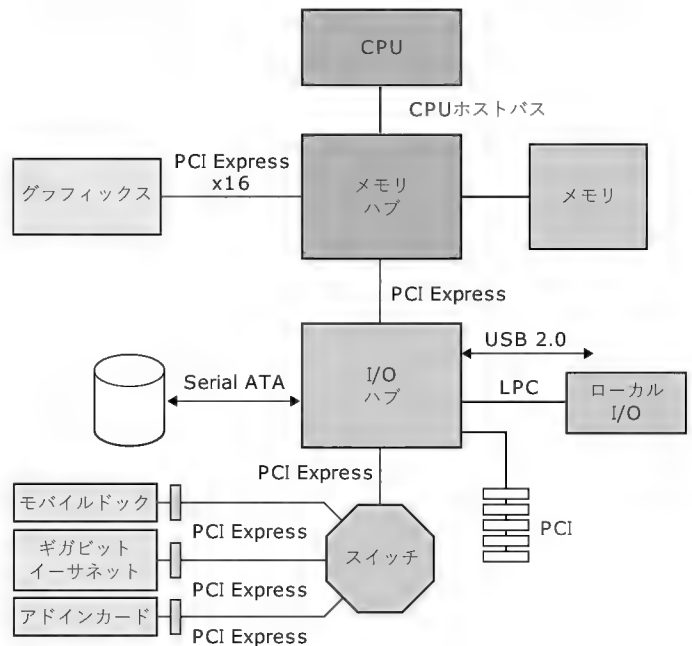
1. PCI Expressの概要

従来のPCIシステムおよびPCI Expressシステムの例を**図1**，実際に想定されるPCI Expressプラットホーム例を**図2**に示す．従来のPCI, PCI-X, AGP, Hub Linkといったバスが PCI Expressで置き換わり，既存のPCI/PCI-Xデバイスを接続するためにブリッジが使用される．チップセット間の接続もPCI Expressとなり，IEEE1394, Serial ATA, USB2.0などはI/Oハブにより PCI Expressに接続される．またサーバにおいては，InfiniBand HCA (Host Channel Adapter)をPCI Expressに接続することも想定される．

PCI Expressをバスとしてみた場合，次のような特徴がある．

- ポイントツーポイントのデュアルシンプレックス(送受信が独立)通信チャネル
- 低電圧差動信号
- エンベデッドクロック(データにクロック信号を重畳)
- スケーラブルな周波数(第1世代で2.5Gbps)とバス幅(1, 2, 4, 8, 12, 16, 32レーン)

注2：PCI ExpressはPCI-SIGのトレードマークである．

〔図2〕デスクトップ/モバイルでのPCI Expressプラットホームの例

またプロトコルとしてみた場合は，以下のような特徴がある．

- 階層化構造
- PCIと同様のロード/ストアアーキテクチャ
- 完全にパケット化されたスプリットトランザクション

● PCI Expressの構成要素

▶ポート(Port)/レーン(Lane)/リンク(Link)(図3)

ポートは物理的には同一半導体内にありリンクを形成するトランスミッタ/レシーバの集合で，論理的にはコンポーネント-リンク間のインターフェースを意味する．

レーンは差動信号ペアのセットで，送信側の信号ペア，受信側の信号ペアからなる．

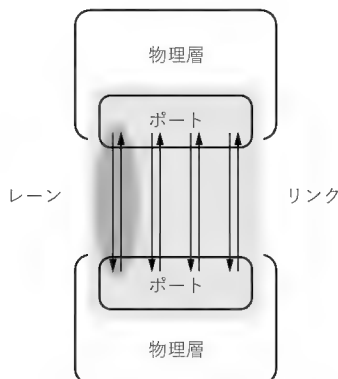
リンクは二つのポートとその間を結ぶレーンの集まりであり，コンポーネント間のデュアルシンプレックス通信パスである．「×Nリンク」注3はN本のレーンから構成され，現在の規格ではN=1/2/4/8/12/16/32が定義されている．レーン幅を可変することにより，スケーラブルなバンド幅を構成することが可能である（**表1**）．

▶ルートコンプレックス（Root Complex）

I/O構造の最上位に位置し，CPUやメモリサブシステムをI/Oに接続する．ブロック図などではメモリハブと記述されていることが多い．ルートコンプレックスは一つ以上のPCI Expressポート（ルートポート）をもち，それぞれのポートは独立したI/O階層ドメインを形成する．I/O階層ドメインは，単純なエンドポイントである場合や，多数のスイッチやエンドポイントから形成される場合がある．

▶エンドポイント（End Point）

タイプ00hのコンフィグレーション空間ヘッダをもつデバイス（具体的にはブリッジ以外のデバイス）で，レガシーエンドポイントとPCI Expressエンドポイントに分けられる．両者の大きな違いは，PCI ExpressエンドポイントはBAR（ベースアドレスレジスタ）でI/Oリソースを要求せず，このためI/Oリクエストを生成しない．またPCI Expressエンドポイントはロックリクエストもサポートしていない．

〔図3〕ポート/レーン/リンクの関係

〔表1〕代表的なリンク構成と伝送バンド幅

リンク構成	信号レートバンド幅（片方向，bps）	実効伝送バンド幅（バイト/s）	
		片方向	双方向
×1リンク	2.5G	250M	500M
×4リンク	10G	1G	2G
×8リンク	20G	2G	4G
×16リンク	40G	4G	8G
×32リンク	80G	8G	16G

注：データは8B10Bコーディングされるため，実効伝送バンド幅は信号レートバンド幅の80%となる

▶スイッチ（Switch）

二つ以上のポートを結合し，ポート間でのパケットルーティングを行う．コンフィグレーションソフトウェアからは，スイッチは仮想PCI-PCIブリッジの集合体と認識される（**図4**）．

▶PCI Express-PCIブリッジ（PCI Express-PCI Bridge）

PCI ExpressからPCI/PCI-Xへの接続を提供する．これにより既存のPCI/PCI-XデバイスをPCI Expressシステム上で使用することができる．

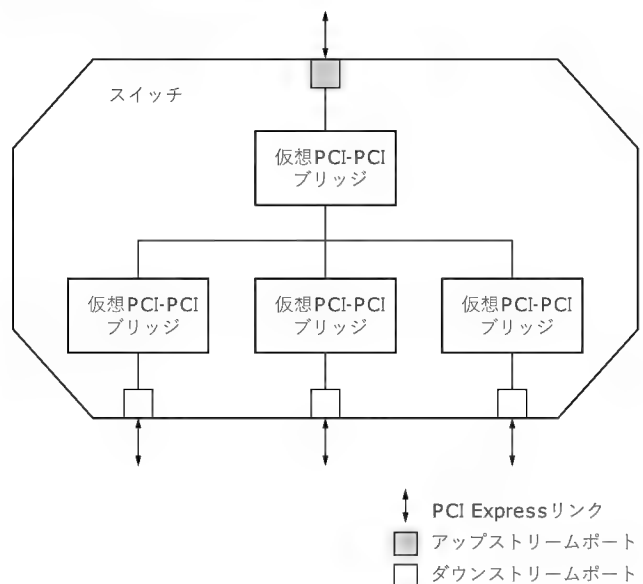
● 階層アーキテクチャ

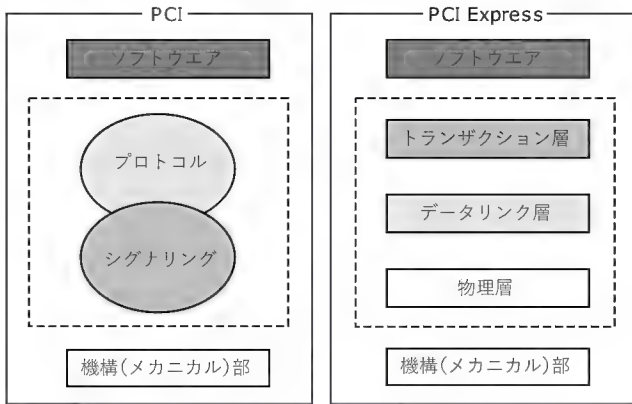
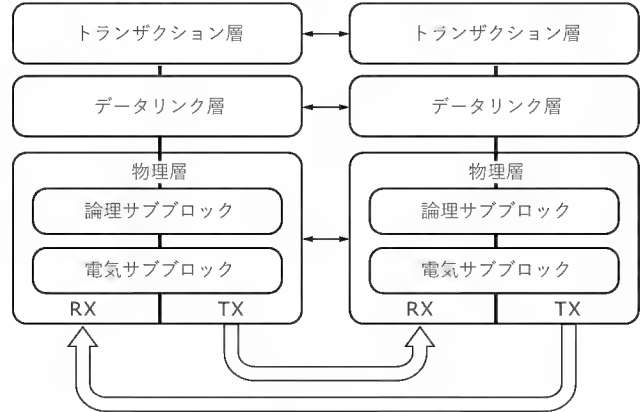
従来のPCIのアーキテクチャは，プロトコルとシグナリングが密接に関連する構造だったが，PCI Expressでは一般的な通信プロトコルやInfiniBandのように，独立した階層構造となっている（**図5**）．これにより各層のモジュール性が確保され，スケーラビリティをもたせることやモジュールの再利用が可能となる．たとえば，新たな信号コーディング方式や伝送媒体を採用する場合，物理層を変更するだけでデータリンク層やトランザクション層は変更せずに対応できる．

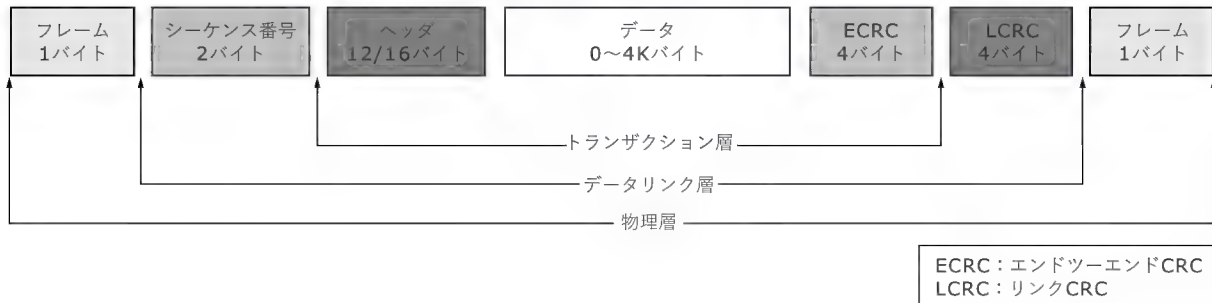
アーキテクチャの中心となるのはトランザクション層，データリンク層，物理層で，それぞれ次のような役割をもつ（**図6**）．

▶トランザクション層

トランザクション層は最上位に位置し，トランザクションレイヤパケット（TLP）の組み立て，分解機能をもつ．TLPはリードやライト，各種イベントといったトランザクションの伝達に用いられる．またトランザクション層はTLPのためのクレジットを用いたフロー制御を行う．各層におけるTLPの概要を**図7**に示す．

〔図4〕スイッチの論理的構造

注3：一般的な読み方は「バイN」．数字の読み方は，筆者の場合N=8までは英語読み，12以上は日本語読みである．

〔図5〕PCI と PCI Express のアーキテクチャ

〔図6〕PCI Express の階層構造

〔図7〕トランザクション層パケット(TLP)フォーマット

ECRC：エンドツーエンドCRC
LCRC：リンクCRC

▶**データリンク層**

データリンク層のおもな役割は，エラー検出/訂正(再送)によりTLPのデータ完全性を保証することと，リンク管理である．データリンク層間ではリンク管理やフロー制御のためのパケットをやりとりする．このパケットはTLPと区別するために，データリンクレイヤパケット(DLLP)と呼ばれる．

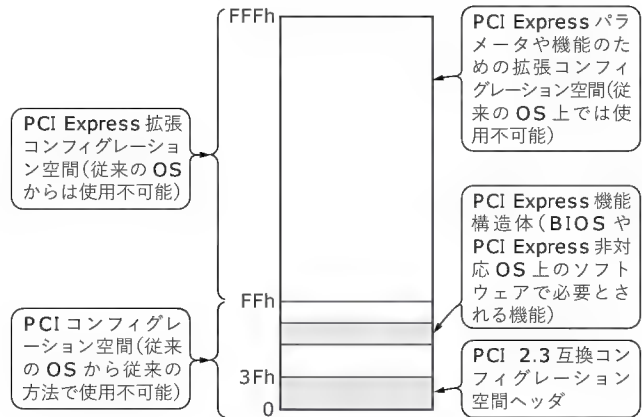
▶**物理層**

物理層はドライバ，入力バッファ，SerDes[注4]，PLLやインピーダンス整合回路といったインターフェース動作のために必要な回路を含んでいる．また論理的な機能としてインターフェースの初期化・保守の機能をもつ．物理層はデータリンク層/トランザクション層を実際のリンクで使用される信号技術から独立させる役目ももっている．

各層についての詳細は後述する．

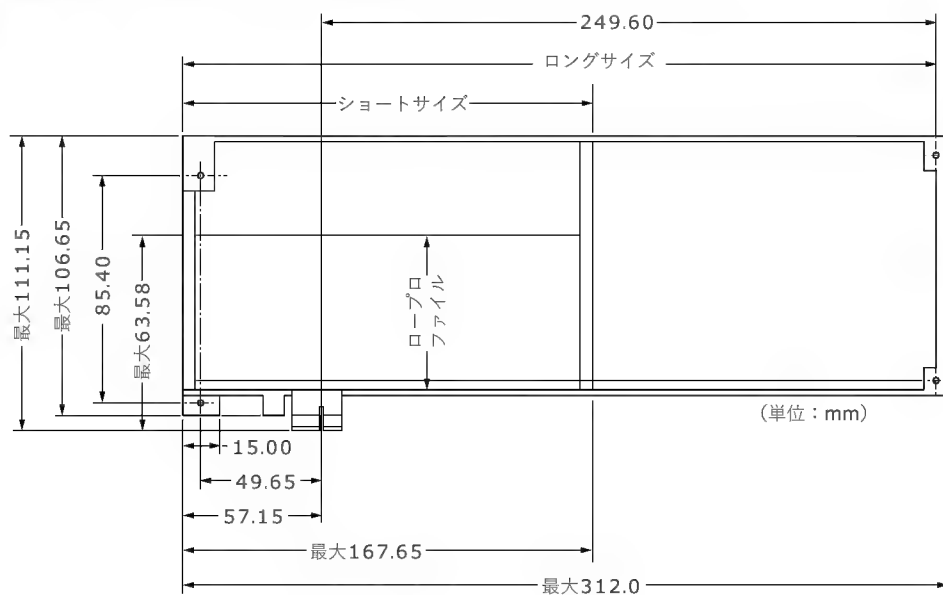
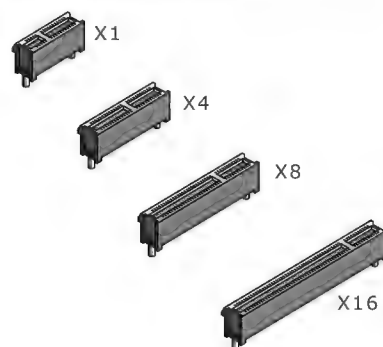
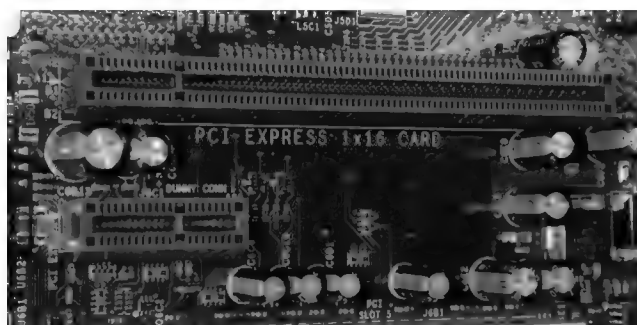
● コンフィグレーション空間

PCI Expressは従来のPCIと同様にコンフィグレーション空間をもつが，その大きさは従来のPCIが256バイトであるのに対し，4096バイトへと拡張されている(**図8**)．これにより，多数のデバイス固有レジスタセットを必要とするデバイス(ホストブリッジなど)に対しても，将来的に十分な空間が確保されている．

〔図8〕PCI Express のコンフィグレーション空間

PCI Expressでは，コンフィグレーション空間へのアクセスはフラットなメモリ空間へのアクセス(コンフィグレーションリード/ライト)で行われ，バス/デバイス/機能/レジスタ番号はメモリアドレスにマップされている．

空間の先頭256バイトはPCIコンフィグレーション空間として，BIOSや従来のOSからI/OポートCF8/CFCを使用した方

注4：Serializer/Deserializerの略で，パラレル-シリアル/シリアル-パラレル変換器のこと．

〔図9〕PCI Expressアドインカードの形状

〔図10〕PCI Expressのコネクタ形状

〔写真1〕PCI Expressのコネクタ(上：×16，下：×1)

法でもアクセスできる．従来のアクセスをPCI Expressでのアクセスに変換する機能はホストブリッジ上に実装される．00hから3FhまではPCI 2.3互換のコングレーションヘッダとなっている．これにより，PCI Expressで拡張された機能以外であれば従来のOSやソフトウェアをそのまま使用することができる．PCI Expressで拡張された機能を使用するためには，OSやソフトウェアの修正や新規開発が必要となる．

2. PCI Expressのフォームファクタ

PCI Expressではさまざまなフォームファクタ(形状)が考えられているが，ここでは現在具体化しているものについて紹介する．

● アドインカードとコネクタ

カードの形状としては従来のPCIと同じで，既存のATXシャーシをそのまま使用することが可能である(**図9**)．コネクタはPCIと同じようなもので，コストのかからないものとなっている．リンクとしては×1，×4，×8，×16が規定されており，それぞれコネクタの大きさが異なる(**図10**，**写真1**)．×1の場合は36ピン，×16の場合は164ピンとなっている．コネクタへはリンク幅が同じか狭いカードを装着することができる(**表2**)．

電源としては+3.3V，+12V，+3.3Vaux(オプション)の3種類が供給される(**表3**)．PCIと比較すると消費電力の大きなアドインカードのために+12Vの容量が強化され，+5Vと−12Vが取り除かれている．グラフィックスカード向け×16コネクタでは，当初40Wであった最大消費電力が，近い将来

〔表2〕カードとスロットの互換性

スロット ＼ カード	×1	×4	×8	×16
×1	必須	必須	必須	必須
×4	不可能	必須	可能	可能
×8	不可能	不可能	必須	可能
×16	不可能	不可能	不可能	必須

広いリンク幅のカードを狭いリンク幅のコネクタに装着することは物理的に不可能となっている

〔表3〕コネクタから供給される電源と最大消費電力

コネクタ種別と最大消費電力	×1(デスクトップ) 10W	×1(サーバI/O) ×4/×8/×16 25W	×16(グラフィックス) 75W(パワーアップ時は25W)
+3.3V ±9%	最大3A	最大3A	最大3A
+12V ±8%	最大0.5A	最大2.1A	最大5.5A
+3.3Vaux ±9%	最大375mA	最大375mA	最大375mA

グラフィックス用×16コネクタの最大消費電力は当初40Wだったものが，現在は75Wとなっている

〔表4〕PCI Express コネクタ(スロット)のピン配置

ピン番号	サイドB 名称	サイドA 名称
1	＋12V	PRSNT1#
2	＋12V	＋12V
3	＋12V	＋12V
4	GND	GND
5	SMCLK	TCK
6	SMDAT	TDI
7	GND	TDO
8	＋3.3V	TMS
9	TRST#	＋3.3V
10	3.3Vaux	＋3.3V
11	WAKE#	PERST#
キー		
12	RSVD	GND
13	GND	REFCLK＋
14	PETp0	REFCLK－
15	PETn0	GND
16	GND	PERp0
17	PRSNT2#	PERn0
18	GND	GND
×1の場合はここまで		
19	PETp1	RSVD
20	PETn1	GND
21	GND	PERp1
22	GND	PERn1
23	PETp2	GND
24	PETn2	GND
25	GND	PERp2
26	GND	PERn2
27	PETp3	GND
28	PETn3	GND
29	GND	PERp3
30	RSVD	PERn3
31	PRSNT2#	GND
32	GND	RSVD
×4の場合はここまで		
33	PETp4	RSVD
34	PETn4	GND
35	GND	PERp4
36	GND	PERn4
37	PETp5	GND
38	PETn5	GND
39	GND	PERp5
40	GND	PERn5
41	PETp6	GND
42	PETn6	GND
43	GND	PERp6
44	GND	PERn6
45	PETp7	GND
46	PETn7	GND
47	GND	PERp7
48	PRSNT2#	PERn7
49	GND	GND
×8の場合はここまで		
50	PETp8	RSVD
51	PETn8	GND
52	GND	PERp8
53	GND	PERn8
54	PETp9	GND
55	PETn9	GND
56	GND	PERp9
57	GND	PERn9
58	PETp10	GND
59	PETn10	GND
60	GND	PERp10
61	GND	PERn10
62	PETp11	GND
63	PETn11	GND
64	GND	PERp11
65	GND	PERn11
66	PETp12	GND
67	PETn12	GND
68	GND	PERp12
69	GND	PERn12
70	PETp13	GND
71	PETn13	GND
72	GND	PERp13
73	GND	PERn13
74	PETp14	GND
75	PETn14	GND
76	GND	PERp14
77	GND	PERn14
78	PETp15	GND
79	PETn15	GND
80	GND	PERp15
81	PRSNT2#	PERn15
82	RSVD	GND
×16の場合はここまで		

予想される消費電力増加にともない，75Wに変更されている．

PCI Expressのリンクや電源以外には，リファレンスクロック，カード検出信号，リセット，Wake，オプションでJTAGとSMバスといった信号が接続される．**表4**にピン配置を示す．

最初に実現されるPCI Express対応マザーボードの形態としては，グラフィックス用に×16，汎用I/O用に×1コネクタを搭載，これに従来のPCIコネクタを複数搭載したものになると思われる[注5]．

● プラグインカード(NEWCARD)

現在NEWCARD[注6]と呼ばれていて，PCカードを置き換えるものである．大きさはシングルでPCカードの半分程度，ダブルはシングルを横に2枚並べた大きさとなる(**図11**)．二つのスロットが横方向に並ぶこととなり，NEWCARD2枚のスロットで現在のPCカード1枚のスロットとほぼ同じ大きさとなる．これにより，薄型化するモバイルプラットホームに対応することができる．スロットからはイジェクト機構が省略され，コネクタも簡単な構造になるなど，低コストを意識したものとなっている．

〔図11〕NEWCARDの外観例(シングル幅)

コネクタは片側に28のコンタクトが1mmピッチで並んだ形となっていて，抜き差しの回数としては5000回から1万回の耐久性をもつ．NEWCARDスロットにはインターフェースとしてPCI Express(×1リンク)，USB2.0が直接出ており，NEWCARDはPCI Expressを使用したカード，USB 2.0を使用したカードとなる(**図12**)．

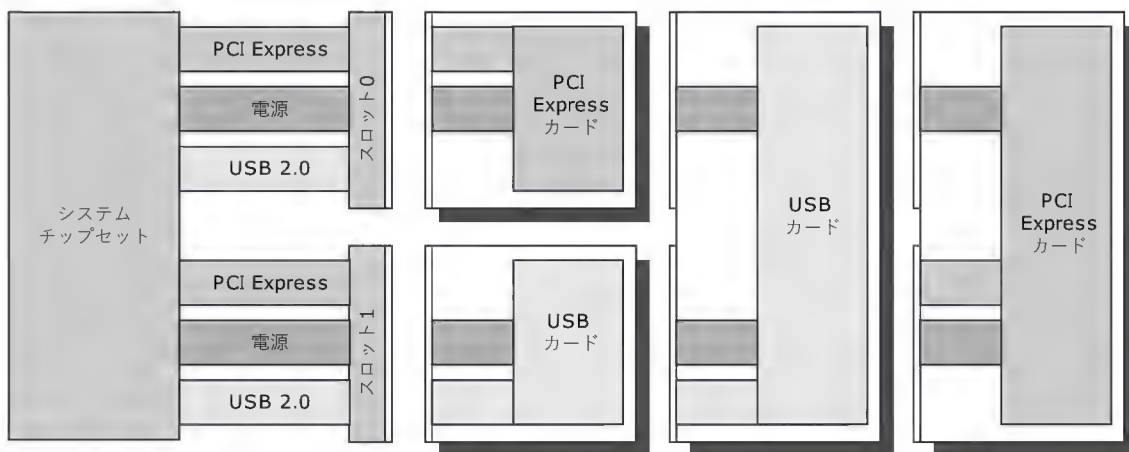
NEWCARDの規格は現在ドラフトレビューされており，2003年末までにはPCMCIAからリリースされる予定である．

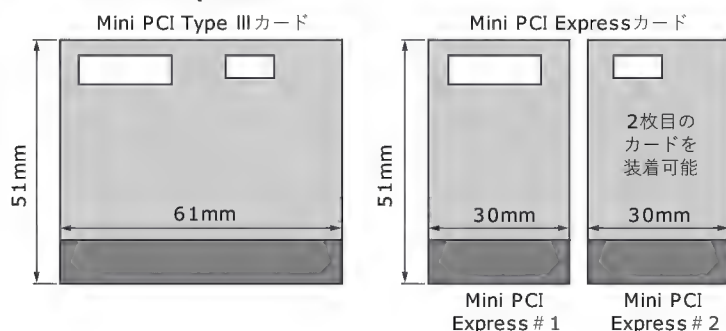
● Mini PCI Express

現在のMini PCIを置き換えるもので，エンドユーザーによる着脱ではなく，BTO/CTO(Build To Order/Configure To Order)で製造時に装着するといった状況を想定している．カードの大きさはMini PCI Type IIIカードの半分となっていて，Mini PCIと同じスペースで52ピンのモジュールを二つ装着することが可能となっている(**図13**)．

インターフェースとしては，NEWCARDと同じようにPCI Express(×1リンク)とUSB2.0が直接出ている．これによりMini PCI ExpressとNEWCARD間で共通の技術が使用でき，相互の移行が容易となる．

注5：ISA/PCIの共存したマザーボードのようなものを想像してほしい．
注6：現在はNEWCARDと呼ばれているが，名称が変更となる可能性がある．

〔図12〕NEWCARDスロットの構成例

注：図では省略されているが，SMバスも各スロットに接続されている

〔図13〕Mini PCI Expressカード

Mini PCI Expressの規格は現在レビジョン1.0のドラフトレビューがPCI-SIGメンバにより実施されている．

● サーバI/Oモジュール（SIOM）

サーバやワークステーションでの使用を想定したもので，現在PCI-SIGで検討が進んでいる．ホストインターフェースとしては×1から×16までのリンクの使用が想定されている．カートリッジのような形[注7]で扱いやすく，シャーシを開閉することなくホットリムーブ/ホットプラグを行うことができる．

注7：InfiniBandのモジュールに非常によく似ている．

3. PCI Expressのアーキテクチャ

PCI Expressアーキテクチャの中心となっているトランザクション層，データリンク層，物理層について，それぞれポイントとなる点を解説する．

● トランザクション層

トランザクション層のおもな役割は，上位のソフトウェア層と下位のデータリンク層の間でトランザクションレイヤパケット（TLP）の組み立てと分解を行うことである．

▶アドレス空間とトランザクションタイプ

PCI Expressでは四つのアドレス空間が定義されている．従来のPCIでサポートされていたメモリ，I/O，コンフィグレー

Column 1
PCI ExpressとInfiniBand

筆者は以前，2001年11月号の本誌でInfiniBandについて紹介した．当時PCI Expressはまだ3GIOと呼ばれており，その概要しかわかっていなかった．InfiniBandの規格を見て感じたことは，上位層であるトランスポート層や通信/ネットワーク管理といったものが細かに規定されており，「重いプロトコル」だな，ということであった（ちなみにInfiniBandのアーキテクチャ仕様書は約2000ページある）．InfiniBandはストレージ向けで3GIOとは直接バッティングしないが，InfiniBandのサブセットであれば競合すると思っていた．ふたを開けてみると，PCI ExpressはInfiniBandのぜい肉をそぎ落として，既存のPCIとの互換性を最大限考慮したものとなっていた．物理層やデータリンク層については，InfiniBandとほとんど同じものとなっている．

InfiniBandはその後Intelが対応半導体の開発を中止，MicrosoftもWindows .NET Server（Windows Server 2003）での対応を中止するなど，以前のような勢いがなくなってきた．これはInfiniBandがまったく新しい，革命的（revolutionary）アプローチであったのに加え，その仕様も複雑で開発にも工数がかかったのに対し，各メーカーは既存の技術を生かした進化的（Evolutionary）アプローチに注力したためだと考えられる．

現在InfiniBandは仕様の1.1がリリースされ，対応製品も一部メーカーから出荷されている．またInfiniBandをベースとした独自I/Oをもった製品を開発販売しているメーカーもある．

ション空間に加えて，メッセージ空間が追加された[注8]．それぞれの空間に対してトランザクションタイプが定義されている（**表5**）．メモリトランザクションでは32ビットアドレスと64ビットアドレスが使用できる．I/OトランザクションはI/O空間を使用するレガシーデバイスの互換性のためにサポートされており，将来的には必要性がなくなってくる（PCI ExpressではI/O空間の使用は推奨されていない）．メッセージトランザクションは単にメッセージとも呼ばれ，PCI Expressデバイス間のインバンドでのイベント通知やメッセージ交換に使用される．割り込み要求や確認はメッセージを「仮想ワイヤ」として使用することにより伝達される．

〔表5〕アドレス空間とトランザクションタイプ

アドレス空間	トランザクションタイプ	Posted/Non-posted	用途
メモリ	Read	NP	メモリ空間とのデータ転送
	Write	P	
I/O	Read	NP	I/O空間とのデータ転送
	Write	NP	
コンフィグレーション	Read	NP	デバイスのコンフィグレーションとセットアップ
	Write	NP	
メッセージ	基本（ベンダ定義を含む）	P	イベント通知や一般的なメッセージ送信

Non-posted（NP）の場合は完了パケットの返送が必要，Postedの場合は不要

▶トランザクションレイヤパケット（TLP）

TLPのフォーマットを**図7**（p.83）に示す．ヘッダ長は3DW（DWはダブルワードの略で，合計12バイト）または4DW（16バイト）で，TLPのフォーマット（ヘッダ長とペイロードの有無），トランザクションタイプ，トラフィッククラス（TC），アトリビュートやペイロード長などの情報が含まれる．パケット内の最大ペイロード長は1024DW（4096バイト）である．

ECRCはエンドツーエンドのデータ完全性を保証するためのもので，TLP部分の32ビットCRCである．これはスイッチ内部などでTLPにエラーが発生した場合，LCRC（リンクCRC）ではエラーを検出できないためである（エラーとなったTLPでLCRCが再計算されるため）．

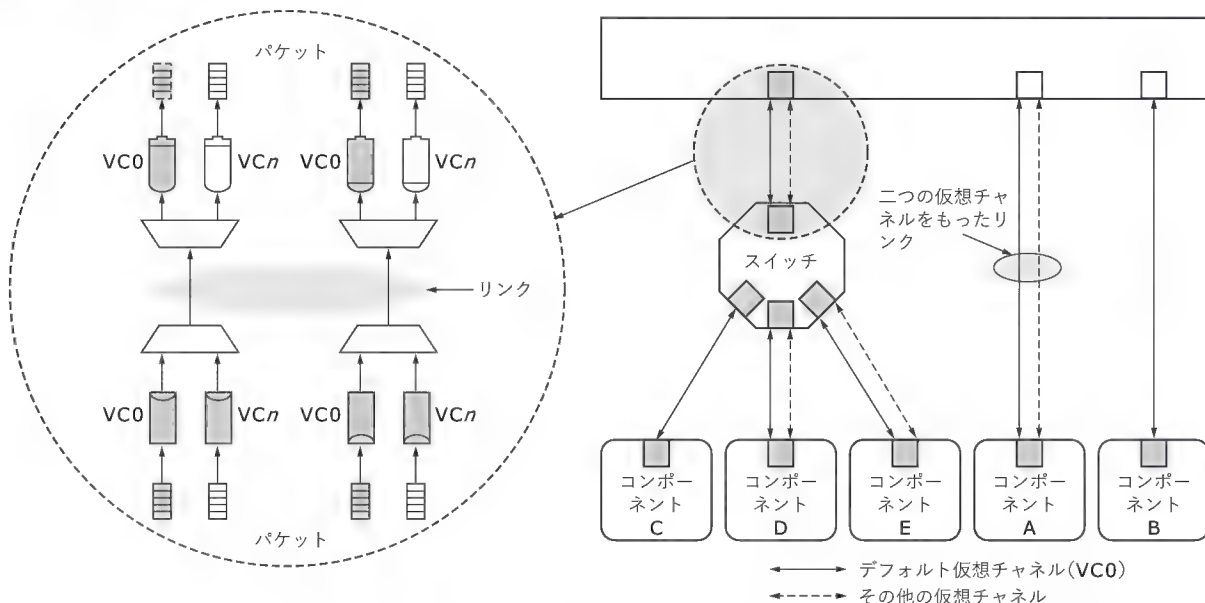
リクエストは完了パケットが不要なもの（Posted）と必要なもの（Non-posted）がある．Non-postedリクエストは，リクエストと完了が1対1で連続しなくてもよいスプリットトランザクション方式で，トランザクションIDによりリクエストと完了が関連づけられる．

▶トラフィッククラスと仮想チャネル

上位のソフトウェアはトラフィッククラス（TC：Traffic Class）を使用することによりトラフィックの差別化を行うことができる．たとえば，映像データをネットワークのデータよりも優先して転送するといったことが可能となる．TCはTC0からTC7までの八つがある．

仮想チャネル（VC：Virtual Channel）はそれぞれ独立した仮想通信パスで，それぞれがリソース（バッファやキュー）をもち，独立したフロー制御を行う（**図14**）．VC0は必須で，コストパフォーマンスのトレードオフに応じてその他の仮想チャネル（VC1～VC7）が実装される．

トランザクション層内ではTCがVCにマッピングされる．一つのVCに対して一つ，または複数のTCをマッピングできる（VCの数が少ない場合）．単純な例では，各TCから各VC

〔図14〕仮想チャネルの概念

注8：メッセージ空間というよりも，メッセージというトランザクションタイプが追加されたというほうが自然である．

に1対1，すべてのTCをVC0にマッピングといったことが考えられる．TC0-VC0のマッピングは必須/固定で，それ以外のマッピングは上位のソフトウェアから制御される．

▶**フロー制御**

受信バッファのオーバフローを避け，伝送の順序を確立するためにフロー制御（FC：Flow Control）が行われる．フロー制御はリンク間のポイントツーポイントで行われ，エンドツーエンドではない．したがって，フロー制御により最終的な相手（コンプリータ）にパケットが届いたことを確認することはできない．

PCI Expressでは「クレジット」によるフロー制御を行う．受信側はリンク初期化時にバッファ容量（クレジット値）を送信側に通知し，送信側はクレジット値と送信するパケットの長さを比較して一定の残りがある場合のみパケットを送信する[注9]．クレジットには六つの種類があり，それぞれで「残高管理」が行われる（**表6**）．

フロー制御の情報交換はデータリンク層のDLLPを使用して行われる．フロー制御はTLPにのみ適用され，DLLPには適用されない（DLLPは常時送受信可能）．

● データリンク層

データリンク層のおもな役割は，リンク上の二つのコンポーネント間で信頼性の高いTLP交換機能を提供することである．

▶**TLPの扱い**

トランザクション層から受け取ったTLPに対しては，先頭に2バイト（使用しているのは12ビット）のシーケンス番号，末尾に4バイトのリンクCRC（LCRC）を付加して，物理層に渡される（**図7**，p.83）．TLPはリトライバッファに保管され，相手から受信確認（ACK）が届くまで再送される．TLPの送信に失敗が続いた場合は，リンク異常であると判断して物理層に対してリンクの再トレーニングを要求する．リンクのトレーニングが失敗した場合，データリンクの状態はインアクティブに遷移する．

物理層から受け取ったTLPはシーケンス番号とLCRCが検査され，正常であればトランザクション層に渡される．エラーがあった場合は再送を要求する．

▶**データリンクレイヤパケット（DLLP）**

データリンク層が生成するパケットはDLLPと呼ばれ，データリンク層間でやりとりされる．DLLPには次のような種類がある．

- Ack/Nak
  TLPの受信確認，リトライ（再送）
- InitFC1/InitFC2/UpdateFC
  フロー制御の初期化とアップデート
- 電源管理のためのDLLP

DLLPの長さは6バイトで，種類を示すDLLPタイプ（1バイト），DLLPの種類で固有の情報（3バイト），CRC（2バイト）から構成される（**図15**）．

● 物理層-論理サブブロック

物理層の論理サブブロックでのおもな役割は，データリンク層から受け取ったパケットを電気サブブロックで送信できる形式に変換することである．また物理層を制御/管理する機能ももっている．

▶**データ符号化とパラレル-シリアル変換**

PCI Expressはデータ符号化に8B/10B変換を用いる（**コラム2**を参照）．変換されたデータはシリアル変換され，LSBからレーン上に送信される．レーンが複数ある場合は，符号化の前にデータがバイト単位で各レーンに割り振られる（**図16**）．

▶**特殊符号（Kコード）**

8B/10B符号では通常のデータを表現する符号のほかに，Kコード（Kキャラクタ）と呼ばれる特殊符号があり，TLP/DLLPのフレーミングやリンク管理に使用される．TLPとDLLPは容易に識別できるように，異なる開始フレーミング符号が付加される（TLPはSTP/K27.7，DLLPはSDP/K28.2）．

▶**データスクランブリング**

特殊符号以外のデータについては，リニアフィードバックシフトレジスタ（LFSR）[注10]を用いてデータのスクランブリングを行っている．これにより同一や規則的なデータが連続した場合もスクランブル後のデータはほぼランダムとなり，信号のスペクトルが広がることによりEMIを低減することができる．送信側でのスクランブリングは8B/10B符号化の前に行われ，受信側でのデスクランブリングは8B/10B復号化の後で行われる．

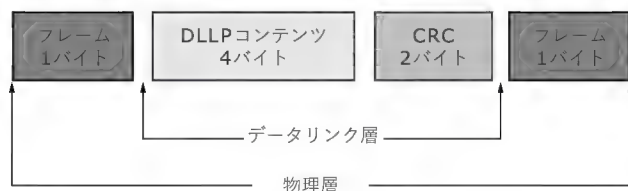
▶**リンク初期化とトレーニング**

リンクは通常に使用される前に，初期化とトレーニングが行われる．トレーニングでは次のようなことが実施される．

- データレートのネゴシエーション

〔表6〕フロー制御のクレジットタイプ

クレジットタイプ	適用されるTLPの情報
PH	Postedリクエスト ヘッダ
PD	Postedリクエスト データペイロード
NPH	Non-postedリクエスト ヘッダ
NPD	Non-postedリクエスト データペイロード
CPLH	コンプリーション ヘッダ
CPLD	コンプリーション データペイロード

〔図15〕データリンク層パケット（DLLP）のフォーマット

注9：PCI Expressワークグループのチェアマン Ajay Bhatt氏は，クレジットカードの利用限度額にたとえていた．

注10：LFSRで使用される多項式が規格の1.0aで変更された．

Column 2

8B/10B符号化

8B/10B符号化はPCI Expressのみでなく100Base-XやシリアルATAでも使用されている符号化方式である．1983年にIBMが開発したもので，論文「A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code」のPDFファイルをIBM JournalのWebサイト（http://www.research.ibm.com/journal/）から入手可能である．

● 符号化の特徴

8B/10B符号化は，8ビットのデータを10ビットの符号に変換する方法で，次のような特徴をもっている．

(1) 0や1が連続するデータでも符号化後はビット変化（信号のエッジ）が多くなり，データからのクロック抽出が可能となる（たとえば00hは0110001011に変換される（**図A**）
(2) 0の数と1の数が同じになるため，DCバランスがとれ，ACカップリングが可能
(3) 符号空間が広がるため，データ以外に制御のための特殊符号を入れることができる
(4) 減衰の大きな1/5以下（<250MHz）の周波数成分を減少させることができる
(5) ディスパリティにより符号のエラー検出が可能
(6) 実効データ転送レートは8ビット/10ビットで80％になってしまう

● 符号化の方法

8ビットデータの各ビットは，LSBからMSBにむかってA，B，C，D，E，F，G，Hと呼ばれ，ABCDE（5ビット）とFGH（3ビット）の二つのグループに分けられる．符号化された10ビットはa，b，c，d，e，i，f，g，h，j（アルファベット順でないことに注意）と呼ばれ，abcdei（6ビット）とfghj（4ビット）の二つのグループに分けられる．

入力の種類（データ/特殊符号），ランニングディスパリティ（後述）の値をパラメータとして，ABCDEがabcdeiに，FGHがfghiに変換される．変換後の最大ランレングス（0/1の最大連続数）は5，0と1の比率は5：5，6：4，4：6のいずれかとなる．

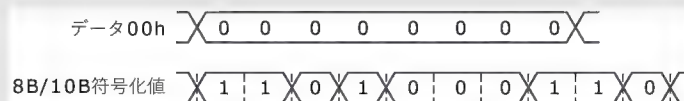
● キャラクタコード

00hからFFhまでのデータはDm.nという名称で呼ばれる．m，nはそれぞれABCDE，FGHの値（10進）の値である．たとえばB4hは10110100bなのでD20.5となる．データはDコードまたはDキャラクタと呼ばれる．

256種類のデータ以外に12種類の特殊符号がKコード（Kキャラクタ）として定義されている（**表A**）．Kコードは各種リンク管理やTLP/DLLPのフレームキャラクタとして用いられる．

〔図A〕データ00hの符号化例

データ00h　0 0 0 0 0 0 0 0

8B/10B符号化値　1 1 0 1 0 0 0 1 1 0

● オーダセット

通常のデータではない制御情報で，Kコード単独またはKコードとDコードの組み合わせが用いられる．PCI Expressではリンクの初期化，トレーニングなどに用いられる．

● ディスパリティ

ディスパリティとは，符号化されたデータ（シンボル）の1の数と0の数の違いである．ディスパリティには次の二つの状態がある．

- ニュートラル：0と1の数が同じ（5個）
- ポジティブ　：1の数が0の数より多い（6個と4個）
- ネガティブ　：0の数が1の数より多い（6個と4個）

ランニングディスパリティ（RD）とは，送信されるシンボルのディスパリティを積算したもので，同様にニュートラル/ポジティブ/ネガティブ状態のいずれかをとる．

8B/10B符号化では，DCバランスを保つために0と1の出現回数を同じにする必要があるため，以下のルールが適用される．

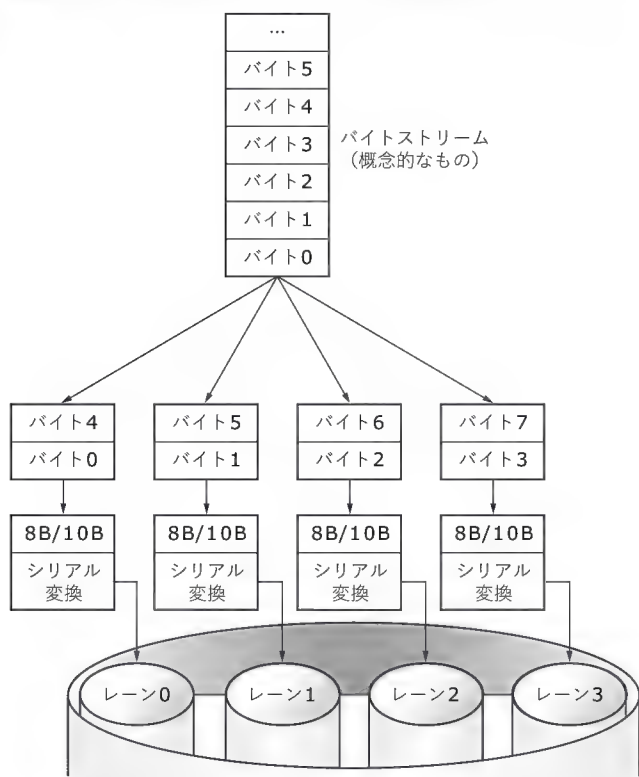
- 現在のRDがネガティブの場合は，次のシンボルのディスパリティはニュートラルかポジティブでなければならない
- 現在のRDがポジティブの場合は，次のシンボルのディスパリティはニュートラルかネガティブでなければならない

D/Kコードでは，それぞれのコードに対して現在のRDがネガティブの場合（Current RD −）とポジティブの場合（Current RD ＋）のシンボルがある．たとえばD0.0（00h）の場合はCurrent RD −が1001110100，RD ＋が0110001011，D12.3（6Ch）の場合はRD −が0011011100，RD ＋が0011010011である．

受信側では現在のRDから次に受信するシンボルのディスパリティ（Current RD −/RD ＋）を予測でき，伝送路上でのエラーを検出可能である（受信したものが予測と異なる場合）．

〔表A〕12種類のKコード（PCI Expressでの定義）

コード	シンボル	名　称	用　途
K28.5	COM	Comma	リンク初期化と管理に使用
K27.7	STP	Start TLP	TLP開始
K28.2	SDP	Start DLLP	DLLP開始
K29.7	END	End	TLP/DLLP終了
K30.7	EDB	EnD Bad	Nullified TLP終了
K23.7	PAD	Pad	フレーミングとリンク幅/レーン順序の調停に使用
K28.0	SKP	Skip	ポート間のビットレート差補償に使用
K28.1	FTS	Fast Training Sequence	L0sからL0へ遷移するためのオーダセット内で使用
K28.3	IDL	Idle	電気的アイドルオーダセット内で使用
K28.4			（予約済み）
K28.6			（予約済み）
K28.7			（予約済み）

〔図16〕×4リンクでのバイトストライピング例

●ビット/符号同期
●レーン極性の反転(受信側で極性を検出し，必要であれば反転する)
●リンク内でのレーン順序の決定
●リンク幅(×1～×32)のネゴシエーション
●レーン間のスキュー調整

トレーニングはトレーニングシーケンスオーダセット(TS1/TS2)と呼ばれる16シンボルのデータを繰り返し送受信することにより行われる．

L0sというリンクの低消費電力ステートから通常ステート(L0)に復帰する際には，ファーストトレーニングシーケンス(FTS)という短時間で完了するトレーニングが用いられる．

〔表7〕リンクステート

ステート	状　態	L0復帰にかかる時間
L0	アクティブ(通常)	
L0s	リンクはコモンモード電圧 クロックや主電源はオン	16ns～4μs
L1	リンクはコモンモード電圧 クロックはオフ，主電源はオン	1～数10μs
L2	クロック，主電源ともにオフ 補助電源(Vaux)がある場合は供給	システムに依存

L2からの復帰時間は，電源やPLLの立ち上がり時間などに依存する

▶電源管理とリンクステート

リンクの消費電力を低く抑えるために，L0/L0s/L1/L2というリンクステートが定義されている(**表7**)．L0が通常のモードで，L0sからL2へと低消費電力になるが，L0への復帰にも時間がかかるようになる．ソフトウェアによる電源管理に加えて，アクティブステート電源管理を積極的に行うことにより，消費電力を極力小さくすることが可能である(**図17**)．

● 物理層-電気サブブロック

物理層の電気サブブロックでのおもな役割は，論理サブブロックでシリアル化されたデータをレーン上に送信することと，レーン上のデータを受信して論理サブブロックに渡すことである(**図18**)．おもな送信出力/受信入力の規格を**表8**に示す．

▶基準クロックとスペクトラム拡散クロック(SSC：Spread Spectrum Clock)

リンク上のそれぞれのポートは互いに600ppm以内のデータレートで送信を行わなければならない．言い換えると，ビットレートのクロックソースは±300ppmの許容差に収まらなければいけない．

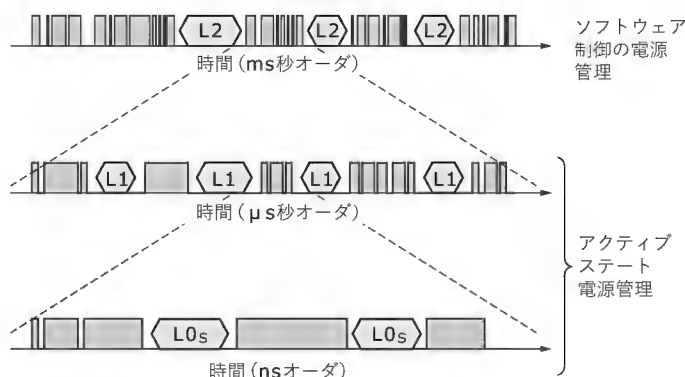
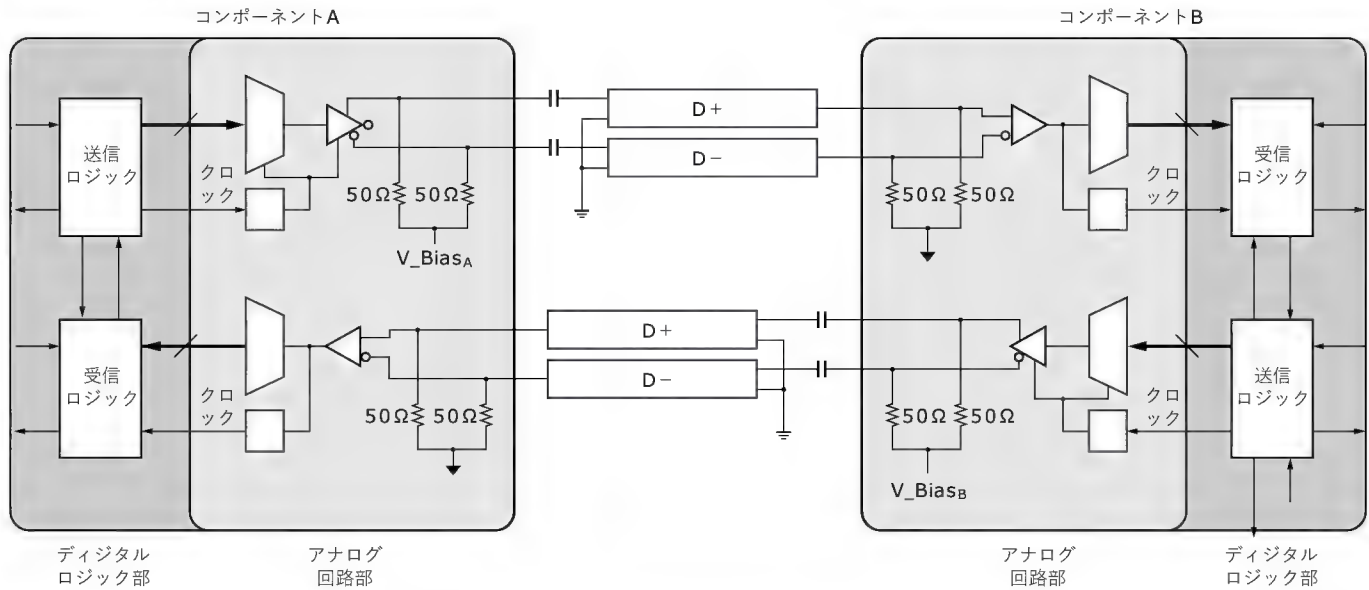
PCI ExpressではEMIを低減するためにデータレートを通常値から＋0%～－0.5%の範囲で変調することができる．変調レートは30kHzから33kHzの範囲に収まらなければいけない．この場合もポート間で600ppmの許容差を満たす必要があるため，SSCを使用する場合は一般的にリンク上の両方のポートは同一クロックソースで動作させる必要がある．

▶ACカップリング

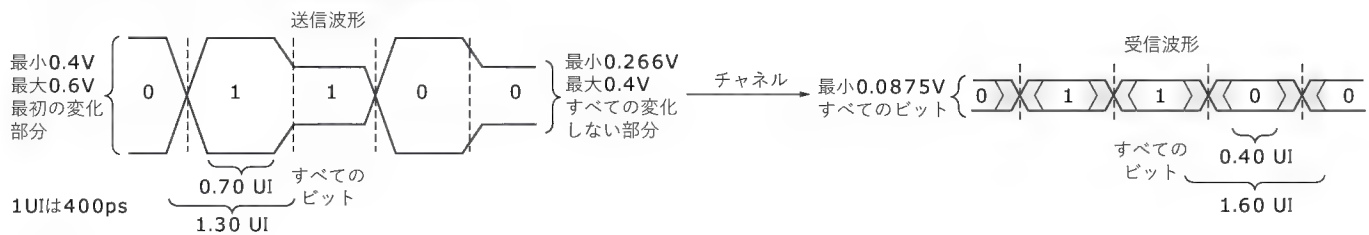
リンクの送信側ではACカップリング用のコンデンサが実装される．これにより，送信側と受信側のDCコモンモード電圧が同一である必要はなくなる．このため送信側と受信側で異なる設計，半導体プロセス，電源電圧を使用することが可能となる．現在のPCIのように，5Vと3.3Vでの相互互換性といったことを考える必要がなくなる．

▶DCコモンモード電圧

受信側のDCコモンモード電圧はACカップリングされているため常に0Vである．

〔図17〕アクティブステート電源管理

〔図18〕物理層の回路例

〔表8〕トランスミッタ出力/レシーバ入力の代表的な規格

パラメータ	最小	標準	最大	単位	
UI（ユニットインターバル）	399.88	400	400.12	ps	400ps ± 300ppm
差動出力電圧	0.800		1.2	V	
差動入力電圧	0.175		1.200	V	
デエンファシス出力	− 3.0	− 3.5	− 4.0	dB	
送信最小アイ幅	0.70			UI	
受信最小アイ幅	0.4			UI	
最大送信ジッタ			0.15	UI	ジッタ中心からの振れ
最大受信ジッタ			0.3	UI	ジッタ中心からの振れ
送信立ち上がり/立ち下がり時間	0.125			UI	時間換算で50ps
差動インピーダンス	80	100	120	Ω	
ACカップリングコンデンサ	75		200	nF	トランスミッタ側

〔図19〕デエンファシスによる送信波形と受信波形

送信側のDCコモンモード電圧は差動ペア間の平均電圧となる．たとえば，差動ペアが0.1Vから0.4Vで振れている場合は，DCコモンモード電圧は0.25Vとなる．

▶電気的アイドル状態

電気的アイドル状態とは，トランスミッタの差動信号ペアの電圧が同一で一定である定常状態のことである．電気的アイドル状態は，おもに節電やインアクティブ状態で使用される．

▶損失（差動電圧振幅の減少）

システム内での損失は，システムが適切に機能するためにコントロールする必要がある重要なパラメータである．最悪の場合での損失はトランスミッタの最小送信差動電圧800mVと最小受信差動電圧175mVより，13.2dBとなる．送信側の差動電圧を大きくすることにより，損失に対して余裕を持たせることができる．4層FR-4基板の場合に想定されるコネクタ含めたレーンの最大レース長は20インチ程度である．

▶デエンファシス

同一極性のビットが連続する場合は，二つ目のビットからは差動電圧レベルを3.5 ± 0.5dB落とす必要がある．これをデエ

〔写真2〕オシロスコープによるPCI Express物理層評価

Intel Developer Forum Japanでの展示．コネクタを経由した受信側のアイパターン測定．信号速度は規格を超える3.25Gbps

ンファシスという(**図19**)．伝送路の周波数依存性減衰のため，変化するビットの場合は高周波成分が多く，減衰により受信側の波形が小さくなるが，変化しないビットの場合は高周波成分が少なく，相対的に受信側の波形が大きくなる．このため，受信側での波形を一定とするためにデエンファシスを行う．

4. PCI Express規格化の現状

Base Specificationは，第2版となるRevision 1.0aが2003年4月15日にリリースされた．軽微な修正や定義の明確化に加えて，プロトコルなどに関する変更も加わっている．またCard Electromechanical Specificationも第2版であるRevision 1.0aも同時にリリースされている．

次の規格については，現在SIGメンバによるレビューが実施されている．

- PCI Express to PCI/PCI-X Bridge Specification Revision 1.0
- Mini PCI Express Card Electromechanical Specification Revision 1.0

また，次のような規格が順次リリースされる予定である．

- PCI Express Server Module Electromechanical Specification
- PCI Express Client Module Specification(PCMCIAから2003年秋にリリース予定)
- PCI Express Advanced TCA Specification(PICMIG)
- PCI Express Advanced Packet Switching Specification (Arapahoe Work Group)

PCI Expressの規格書については，PCI-SIGメンバ企業に所属していればWebより無償でダウンロード可能である．メンバ以外の場合は，PCI-SIGより有償でハードコピーを入手できる．PCI-SIG以外の規格書については，それぞれのWebを参照されたい．

- PCI Express対応製品の開発

国内外の企業によりPCI Express対応の半導体，ボード，コネクタなどの開発が進んでおり，2003年末には最初の対応製品がリリースされると考えられる．またプロトコルアナライザや高性能オシロスコープ/プローブに代表される各種PCI Express対応測定機器は，すでにリリースされているか，または現在開発が進んでいる．

2003年4月に開催されたIntel Developer Forum Japanでは，グラフィックス用に×16，汎用I/O用に×1のPCI Expressを搭載し実際に動作するマザーボードや，各種測定機器が展示されていた(**写真2**)．

- PCI Expressに関する情報について

メンバ企業に所属していれば，PCI-SIGのWeb(http://www.pcisig.org/)から規格書やプレゼンテーション資料を無償でダウンロードすることが可能である．

Intel Developer Network for PCI Expressでは，ユーザー登録(無償)することにより各種ドキュメントをダウンロード，PCI Express関連の情報を電子メールで受け取ることができる(http://developer.intel.com/technology/pciexpress/index.htm)．

PCI Express関連の書籍としては，Intel Pressより「*Introduction to PCI Express:A Hardware and Software Developer's Guide*」が出版されている．

また，PC関連のトレーニング/書籍を提供しているMINDSHARE(http://www.mindshare.com/)はPCI Expressのトレーニングコースをオンサイト/インターネット上で提供しており，PCI Expressを解説した書籍を2003年11月に出版する予定である．

おわりに

PCI Expressでは，すでに5Gbpsや10Gbpsといった速度もロードマップ上では見えてきており，今後10年を越えるレンジで標準I/Oインターコネクトとなることは確実と考えられる．またPCIとの互換性により，従来のISAからPCIよりも早いスピードで移行が進むであろう．

ただし，信号の速度が従来のPCI/PCI-Xよりも格段に速くなっているため，技術的にクリアすべきハードルが多数ある．とくに，電気的部分でのシグナルインテグリティは非常に重要なポイントといえる．

PCI Expressという言葉は一般的になってきたが，現時点で入手できる情報は非常に少ない．本解説が，PCI Expressに対する理解への第一歩になれば幸いである．

さとみ・たかし
アジレント・テクノロジー(株) 電子計測本部
ロジック・高速デジタル　アプリケーション・スペシャリスト

PCI-Xの特徴とプロトコル

村井康秀

前章のPCI Expressは，現在規格が決まったところで，実際の対応製品はこれから登場するという新しい規格である．ここで紹介するPCI-Xは，サーバ用途向けのマザーボードではすでに採用され，PCI-X対応のGigabit EthernetカードやUltraSCSIカードなど，より高速な転送を要求されるカードも市販されているなど，現時点ですぐに利用できる高速バスである．(編集部)

はじめに

現在もパソコンでは標準拡張バスとして利用されているPCIですが，サーバにおいては，転送スピードを上げたPCIの上位互換規格であるPCI-Xが利用されるようになってきました．このPCI-Xは，1999年にPCI-SIG(Peripheral Component Interconnect Special Interest Group)によって策定され，PCIバスのアーキテクチャはそのままに，技術要望を満たした仕様としてデビューしています．さらに，PCI-X 2.0のリリースで高速モードのモード2という，より高速な転送が可能になりました．

ここでは，PCI-Xの概略の特徴や新しく追加されたプロトコル，またPCI-Xモード2の特徴，そして最後に実際のトレースデータを使ってPCI-Xのプロトコルについて解説します．

1. PCI-Xの特徴

● PCIとPCI-Xの性能比較

PCIバスの最高スペックである66MHz/64ビット時のデータ転送レートは533Mバイト/秒ですが，PCI-X 1.0ではこれを133MHz/64ビットにすることで最大1066Mバイト/秒としています．さらに，PCI-X 2.0では同じ133MHz/64ビットで，データ転送レートを2倍(DDR)または4倍(QDR)にしています(**表1**)．PCI-X 2.0では，PCI-X 1.0と互換性のあるモードをモード1，DDRやQDR動作のモードをモード2としています．PCI-X 2.0のモード2は，DDRやQDR動作でさらに信号レベルなども異なる(1.5V)ので，電気的にはまったく別のバスと考えることもできます．

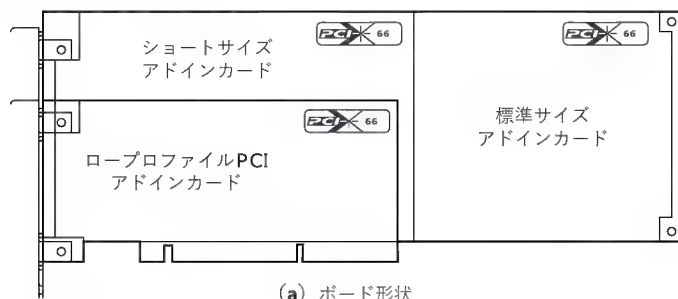
〔**表1**〕**PCIとPCI-Xの性能比較**

規　格	バス幅	クロック周波数	最大データ転送
PCI 2.2	32ビット	33MHz	133Mバイト/秒
PCI 2.2	32ビット	66MHz	266Mバイト/秒
PCI 2.2	64ビット	33MHz	266Mバイト/秒
PCI 2.2	64ビット	66MHz	533Mバイト/秒
PCI-X 1.0	64ビット	66MHz	533Mバイト/秒
PCI-X 1.0	64ビット	133MHz	1,066Mバイト/秒
PCI-X 2.0(DDR)	64ビット	133MHz	2,133Mバイト/秒
PCI-X 2.0(QDR)	64ビット	133MHz	4,266Mバイト/秒

● PCI-X対応カードとシステム

図1にPCI-X対応カードの形状やロゴを示します．PCI-Xには32ビット幅という規格はないので，物理的なボード形状は64ビットPCIカードと同じです．また，電圧キーは3.3V版の位置になります．

また，PCI-X対応システムでは，接続できるスロットの数も増加しています．66MHzのPCIバスでは最大2スロットまでだったのが，66MHzのPCI-Xでは最大4スロットとなってい

〔**図1**〕**PCI-X対応カード**

(a) ボード形状

(b) PCI-X 1.0ロゴ

(c) PCI-X 2.0ロゴ

〔表2〕PCI/PCI-Xシステムのスロット数

規　格	クロック周波数	最大スロット数
PCI	33MHz	4～6
PCI	66MHz	2
PCI-X	66MHz	4
PCI-X	100MHz	2
PCI-X	133MHz	1

〔表4〕システム初期化時の信号

	PCI		PCI-X		
	32ビット	64ビット	66MHz (50～66MHz)	100MHz (66～100MHz)	133MHz (100～133MHz)
TRDY#	"H"	"H"	"L"	"H"	"L"
STOP#	"H"	"H"	"H"	"L"	"L"
REQ64#	"H"	"L"	"L"	"L"	"L"

ます(**表2**).

表2を見ると，クロック周波数が100MHzという項目が見えます．これは，バスクロック100MHz動作のPCI-X対応ボードという仕様があるわけではありません(**図1**にもPCI-X100というロゴはない)．**表2**を見るとわかるように，133MHz動作のPCI-Xの場合，拡張スロットは一つしか実装できません．

66MHzより高速な性能が欲しいが，拡張性も考慮して2スロットは欲しいという要求に対応できるよう，バスクロックを100MHzにすることで最大2スロットまで使えるようにしたシステム側の仕様です．このスロットには133MHz対応のボードを差し込むことで，バスクロックは100MHzで動作します．

● PCIとPCI-Xの互換性

表3にPCIとPCI-Xの互換性を示します．たとえば，既存のPCIシステムにPCI-Xのカードを挿した場合は，PCI-XのカードはPCIボードとして利用できます．また，その逆にPCI-XシステムにPCIカードを挿した場合は，そのPCI-Xスロット(およびそのバスに接続されているPCI-XスロットとPCI-Xデバイス)はPCIモードとして動作します．

図2にカードの識別方法を示します．PCIXCAP信号は，サイドBの38番ピンです．従来のPCIでは，グラウンドに接続されている端子です．バスクロック66MHz対応のPCI-Xではプルダウン，133MHz対応のPCI-Xでは開放状態のままにします．PCI-X対応のシステムではこのピンを操作し，接続されているカードがPCI-Xに対応しているかを判定します．

システム側はPCIXCAP信号でカードを識別できますが，カード側はどうでしょうか．**表4**にシステム初期化時の信号を示します．リセット信号が解除されたとき，つまりRST信号の立ち上がりエッジ時に，**表4**に示した各信号の状態を判定することで，カード側はどのモードで動作すればよいのかを判定します．

2. PCI-Xのプロトコル

● リクエスタとコンプリータ

PCIでは，イニシエータ(マスタ)/ターゲットという呼び方をしていましたが，PCI-Xではイニシエータをリクエスタ(Requester)とコンプリータ(Completer)と呼びます．

また，バスコマンドも一部変更になっています(**表5**)．PCI-X

〔表3〕PCIとPCI-Xの互換性

システムボード＼ターゲットボード		現行のPCIカード			PCI-Xカード	
		33MHz (5V I/O)	33MHz (3.3V I/O または汎用)	66MHz (3.3V I/O または汎用)	66MHz (3.3V I/O または汎用)	133MHz (3.3V I/O または汎用)
従来のシステム	33MHz	33MHz (5V I/O)	33MHz (5V I/O)	33MHz (5V I/O)	33MHz (5V I/O)	33MHz (5V I/O)
	66MHz	－	33	66	a) 33 b) 66	a) 33 b) 66
PCI-Xシステム	66MHz	－	33	33	66	66
	100MHz	－	33	a) 33 b) 66	66	100
	133MHz	－	33	a) 33 b) 66	66	133

PCIの33MHz対応は，PCI-Xのシステムやアドインカード作成には必須事項である．一方，66MHzはオプションであるため，設計者が状況に応じて対応させる．たとえば100MHz対応PCI-Xシステムに66MHz PCIカードを搭載した場合，a)のように通常は33MHzで動作する．PCI-Xシステムが66MHzに対応するようになっていれば，b)のように66MHzで動作させることが可能である(太枠内はPCI-Xモード)．

〔図2〕カードの識別方法

(a) PCIXCAP＝無接続(133MHz対応PCI-X)
(b) PCIXCAP＝プルダウン(66MHz対応PCI-X)
(c) PCIXCAP＝グラウンド(PCI)

で変更になったバスコマンドについては後述します．なお，**表5**の中にある「Alias to Memory Read Block」と「Alias to Memory Write Block」は将来の拡張用で，現在は使えません．

● 追加された機能

PCI-XはPCIと比較して，次の六つの項目が追加/拡張されています．

(1) アトリビュートフェーズ(Attribute Phase)
(2) スプリットトランザクション(Split Transactions)
(3) リラックスオーダリング(Relaxed Ordering)
(4) ノンキャッシュコヒーレントトランザクション(Non-Cache-Coherent Transaction)
(5) オプティマイズウェイトステート(Optimized Wait States)
(6) スタンダードブロックサイズムーブメント(Standard Block Size Movement)

● アトリビュートフェーズ

PCIのターゲット側は，どのイニシエータからのアクセスなのか，バースト転送の場合は何ワードのアクセスをするつもりなのかを判定することはできませんでした．つまり，どこの誰に指示されたかはよくわからないが，とにかく指定されたアドレスが自分のアドレスであればアクセスを行い，FRAME#がディアサートされるのを今か今かと待ちながらバースト転送をしていました．

PCI-Xでは，どこ(バス番号)の誰(デバイス/ファンクション番号)が要求したアクセスなのか，またバースト転送の場合は何バイトのアクセスなのかを，トランザクション開始の時点で判定できるよう，バスアクセスの属性情報が追加されました．

図3に，PCIとPCI-Xのデータ転送のようすを示します．PCIと比較して大きく異なるのは，アドレスフェーズの次のクロックにアトリビュートフェーズが追加された点です．

図4にアトリビュートフェーズの内容を示します．トランザクションサイズやリクエスタのアイデンティティ，またキャッシュ検索要求やリラックスオーダリングの有無などの情報が格納されます．

● スプリットトランザクション

実際のデータ転送効率を良くするためのプロトコルで，データ転送の一連のトランザクションを要求するフェーズと実際に転送されるフェーズを分割するものです．たとえば，リードサイクルにおいて，リクエスタの要求するデータをコンプリータがすぐに用意できない場合，コンプリータはスプリットレスポンスを返すことにより，一度トランザクションを終了します．これによりターゲット側の動作が遅れてもバスは開放されるので，イニシエータ側は転送されるデータを待つことなく，次々と新たな要求をターゲット側に送れます．

また，コンプリータがデータ転送の用意できた時点で，今度はリクエスタとなり，要求元にライトサイクルで返すことによってトランザクションが終了します．

このプロトコルで，不必要なウェイトサイクルやリトライの発生を防止し，実質転送レートを向上させることができます．

〔表5〕PCI-Xのバスコマンド

C/B #	現状のPCIコマンド	PCI-Xコマンド
0110	Memory Read	Memory Read DWORD
0111	Memory Write	Memory Write DWORD
1110	Memory Read Line	Memory Read Block
1111	Memory Write and Inv.	Memory Write Block
1000	Reserved	Alias to Memory Read Block
1001	Reserved	Alias to Memory Write Block
0010	I/O Read	I/O Read
0011	I/O Write	I/O Write
1010	Config Read	Config Read
1011	Config Write	Config Write
1100	Memory Read Multiple	Split Completion
0000	IntAck	IntAck
0001	Special Cycle	Special Cycle
1101	Dual Address Cycle	Dual Address Cycle
0100	Reserved	Reserved
0101	Reserved	Reserved

図5に，スプリットレスポンスのバスの動作を示します．スプリットレスポンスはDEVSEL#とSTOP#を“H”，TRDY#を“L”にして，トランザクションを終了します．

この後，データ転送の準備ができたコンプリータ側はバスの制御権を取得し，リクエスタから要求のあったデータをスプリットコンプレッションコマンドで返します．このときのアドレスフェーズのビット29～8には，スプリットレスポンスを返したときのリクエスタのアトリビュートフェーズの値を入れます．ビット31～30および7は予約されており，ビット6～0はLower Addressを入れます

● リラックスオーダリング

PCIでは，PCI-PCIブリッジは要求を受け取った順番に処理を実行しますが，PCI-Xではこの順番を重要視せず，デバイスが対応できる順番に次々と処理を実行します．つまり，PCIでは複数のデバイスに要求した処理は，必ず要求した順番でしか処理が実行されないため，最初に要求した処理に時間がかかってしまうと，それ以降の処理はずっと待たされるわけです．

PCI-Xでは，後から要求された処理でも先に実行できるので，バスを効率よく使うことが可能です．PCI-X-PCI-Xブリッジがこのように順番を入れ替えることが可能なのは，アトリビュートフェーズにより，リクエスタの情報が伝わっているからです．

リラックスオーダリングを許可する場合は，アトリビュートフェーズのビット29を1にします．

● ノンキャッシュコヒーレントトランザクション

このトランザクションはマルチプロセッサシステムを使う際に，ホストPCI-Xブリッジの負担を軽くしてより高速にデータ転送を実行するためのものです．通常のシングルプロセッサシステムでは，このトランザクションは使いません．

マルチプロセッサシステムにおけるPCIでは，メインメモリ

〔図3〕PCIとPCI-Xのデータ転送

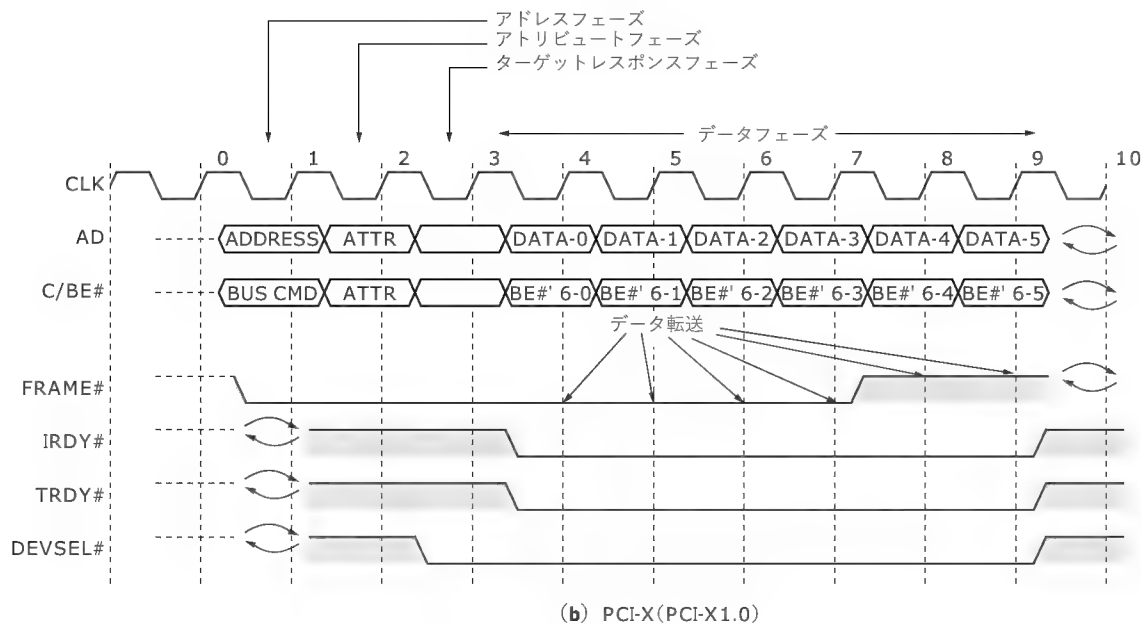
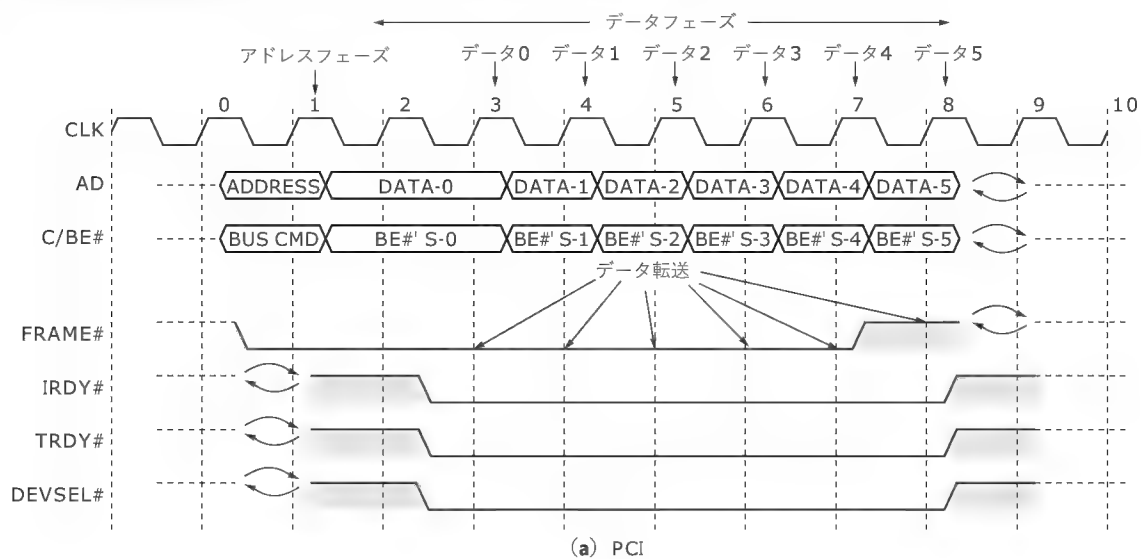
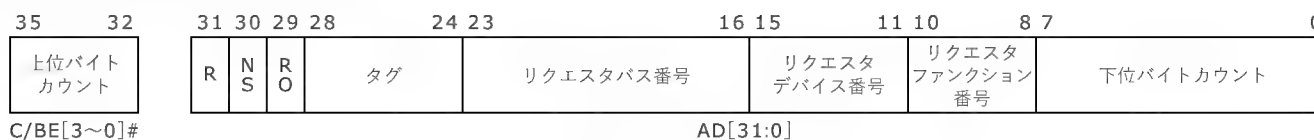
（a）PCI

（b）PCI-X（PCI-X1.0）

〔図4〕アトリビュートフェーズの内容

35–32	31	30	29	28–24	23–16	15–11	10–8	7–0
上位バイトカウント	R	NS	RO	タグ	リクエスタバス番号	リクエスタデバイス番号	リクエスタファンクション番号	下位バイトカウント
C/BE[3～0]#	AD[31:0]							

（a）バースト転送時

35–32	31	30	29	28–24	23–16	15–11	10–8	7–0
バイトイネーブル	R	NS	RO	タグ	リクエスタバス番号	リクエスタデバイス番号	リクエスタファンクション番号	リザーブ
C/BE[3～0]#	AD[31:0]							

（b）シングルデータ転送時

R ：Reserve
NS：No Snoop
RO：Relaxed Ordering

に書き込まれた内容は常にブリッジからキャッシュにアップデートされており，この結果ブリッジがキャッシュにアップデートしている間は，他のトランザクションが起こせませんでした．

PCI-Xでは，このノンキャッシュコヒーレントトランザクションによって，ブリッジがそれぞれのプロセッサのキャッシュをアップデートする作業をキャンセルすることが可能になりました．ゆえに，ブリッジの負担が軽くなり，すぐに次のトランザクションを受け入れられます．

ノンキャッシュコヒーレントトランザクションを使う場合は，アトリビュートフェーズのビット30を1にします．

〔図5〕スプリットレスポンスのバスの動作

● オプティマイズウェイトステート

PCI-Xでは，ウェイトをかけるタイミングにも制限が加えられました．ウェイトを入れられるのは，アトリビュートフェーズの直後から最初のデータ転送を開始するまでの間のみで，一度データ転送が始まると，ウェイトを入れることができません．これはバス転送効率，スループットを上げるためです．

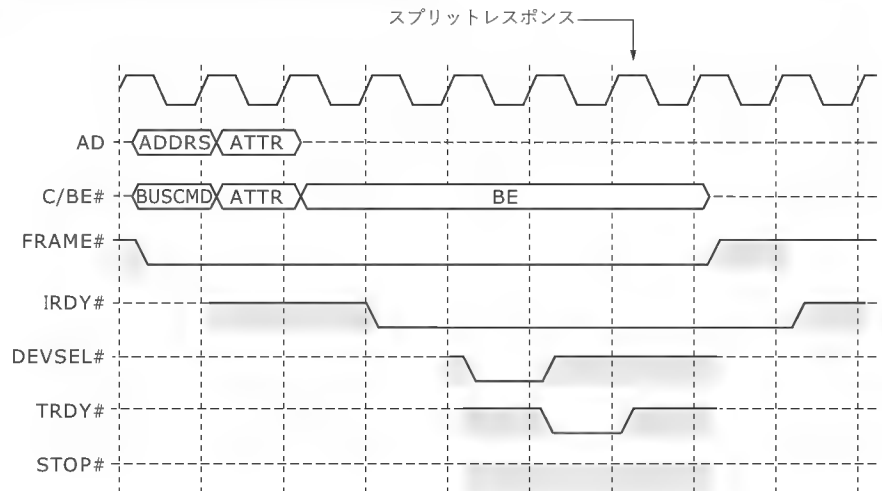
すでに説明したように，PCIではターゲット側が何ワードのバースト転送なのかを判定することはできませんでした．そのため，ターゲット側がデータ転送の準備ができていない場合は，TRDY#をディアサートすることでバースト転送の途中でもイニシエータに対してウェイトを要求できました．

しかし，PCI-Xでは，アトリビュートフェーズにより事前に転送バイト数を知ることができるので，あらかじめ内部のバッファにデータを用意しておくことで，バースト転送の途中でウェイトを要求する必要はなくなりました．

● スタンダードブロックサイズムーブメント

もちろん，何らかの理由で，バースト転送途中でウェイトを入れたい場合もあるでしょう．そのような場合は，128バイト境界においてのみ，そのトランザクションをディスコネクトすることができるという規定もあります．

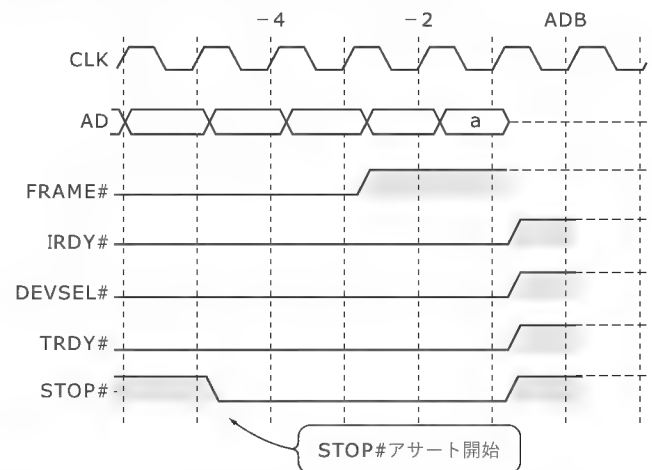
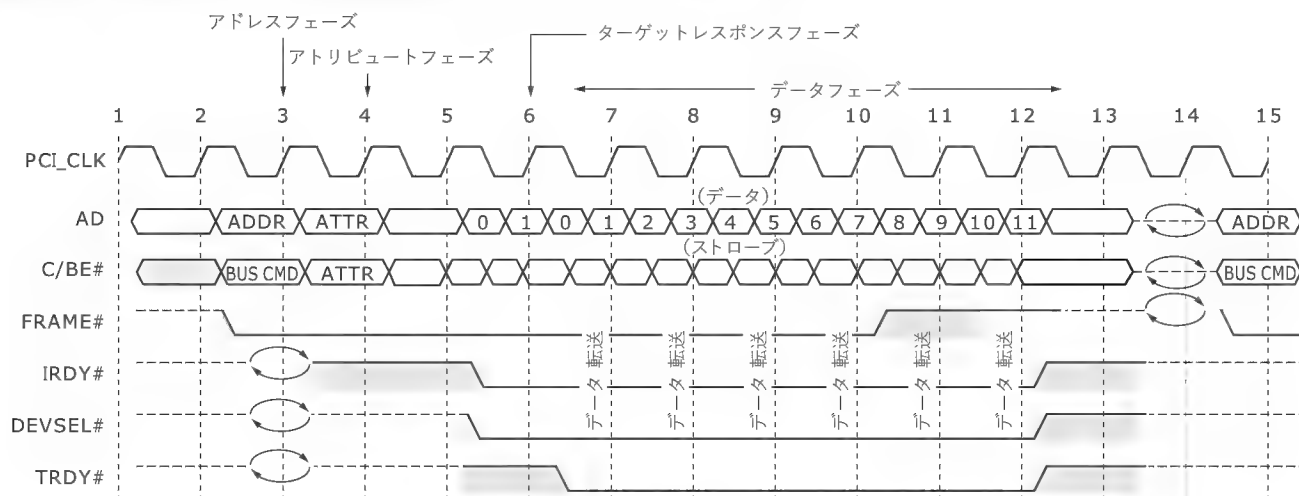
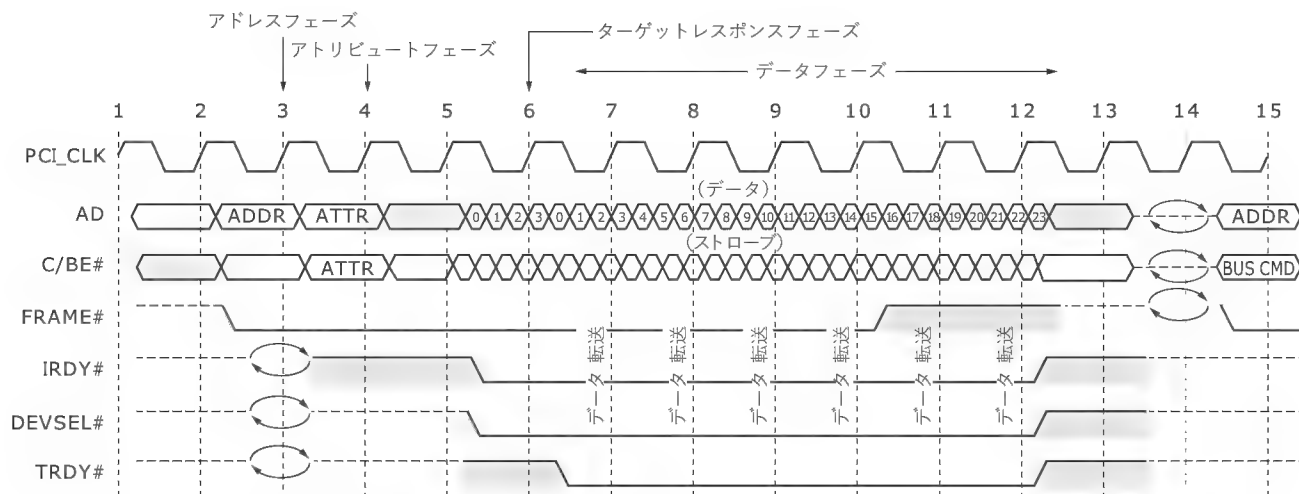
これらの点より，バースト転送対応のデバイスを設計する場合は，内部に最低限128バイト分のバッファは必要であることがわかります．

なお，ディスコネクトの実際の制御方法も，PCI-XではPCIと若干異なります．PCIでは，STOP#のアサート時にTRDY#をアサートするか(データ転送をともなうディスコネクト)，TRDY#はディアサートするか(データ転送をともなわないディスコネクト)などがありました．PCI-Xではオプティマイズウェイトステートにより，データ転送の途中でTRDY#をディアサートすることはできませんし，スタンダードブロックサイズムーブメントによりディスコネクトをかけるバイト境界も規定されています．

したがって，実際の動作は**図6**のようになります．図中の転送データaが128バイト境界の最後のデータ転送だとします．このアローワブルディスコネクトバウンダリ(ADB)の4クロック前までに，STOP#をアサートしておかなければなりません．リクエスタがそれを認識してFRAME#がディアサートされるのはADBの2クロック前となります．

3. PCI-X 2.0について

PCI-X 2.0ではさらなる高速転送を実現するため，PCIやPCI-X 1.0に対して仕様を変更しています．PCI-X 2.0ではPCI-X 1.0と互換性のあるモードをモード1，新しく策定されたモードをモード2と呼びます．

モード2でのデータ転送レートは，DDR(Double Data Rate)で2倍，QDR(Quad Data Rate)で4倍もの高速化を実現しています(**図7**)．このモードを実現するため，システムクロック

〔図6〕アローワブルディスコネクトバウンダリ

〔図7〕PCI-X 2.0モード2でのデータ転送の例

(a) PCI-X266

(b) PCI-X533

QDRの場合，共通クロックの1周期の間に四つのデータを転送する．1組のストローブ信号は共通クロックの2倍の周波数になり，1クロックにつき四つの立ち上がりエッジを提供する．そして，各ストローブ信号は転送の途中で45度オフセットされる

との同期ではなく，ソースシンクロナスクロック(Source Synchronous Clock)を採用して，クロック信号とデータ信号のわずかなスキューをも排除しています．また電圧も，3.3Vから1.5Vに落として高速化に対応しています．

また，実転送レートを向上させるにあたり，プロトコルの改良も行われ，デバイスIDメッセージ(Device ID Message)というコマンドが追加になっています．これによりデバイス間でピアツーピアのデータ転送が可能になりました．

さらに，バス幅を狭めた16ビットインターフェースも規格化されました．標準の64ビットに対して，設計/製造がより容易に行え，転送レートにおいてもDDRやQDRを使えば，それぞれ533Mバイト/秒や1066Mバイト/秒と高速性を保つことが可能となっています．

なお，PCI-X 1.0では**表4**のようにして動作モードを初期化していましたが，モード2ではバスセグメント単位でリセットして再起動しています．

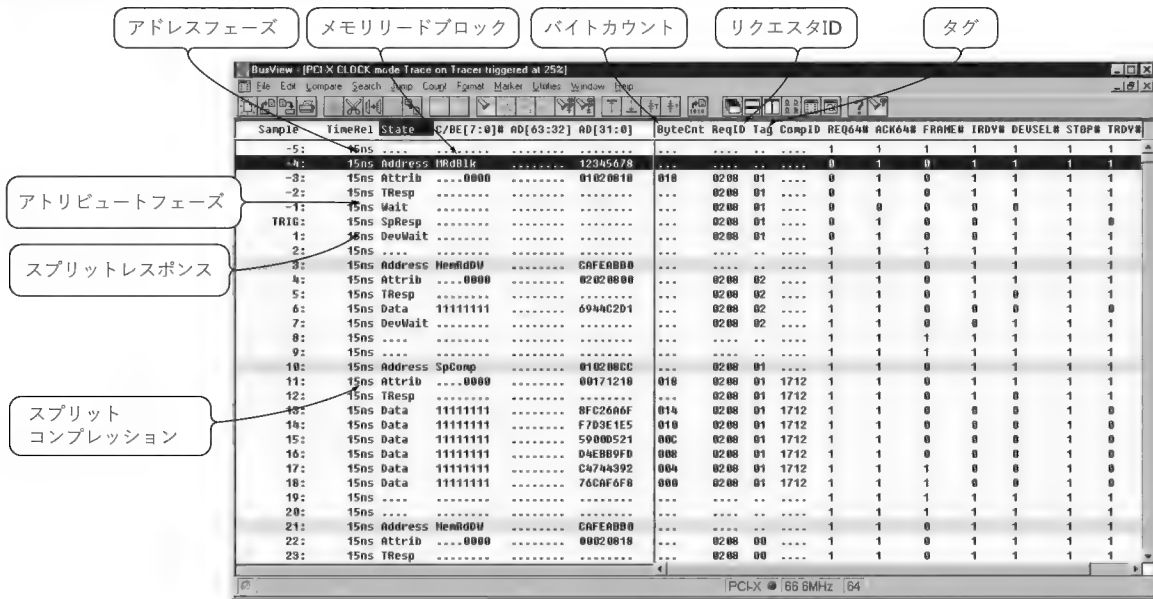
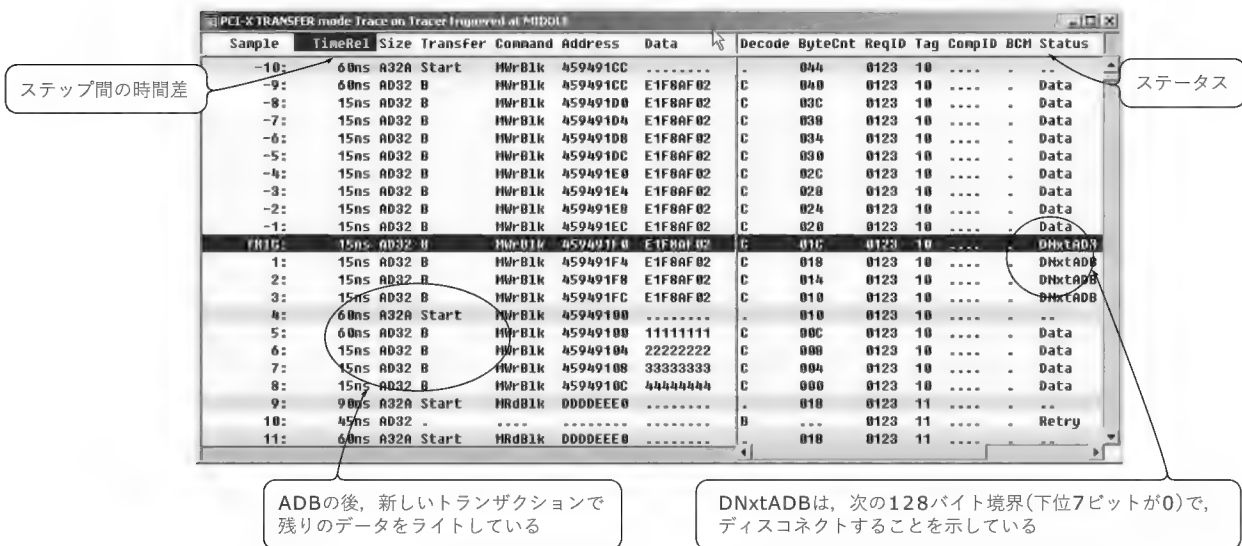
4. PCI-Xの実際のバスの動作例

図8に，実際のトレースデータを示します．これはVMETRO社のPCI-Xアナライザを利用してスプリットレスポンス(SpResp)をトリガに設定してトレースしたデータです．

● Clockモード

トレース方法は，Clockモード(Clockの立ち上がりごとにトレース)としてリスト形式で表示しています〔**図8**(**a**)〕．

(1) Sample「－4」(トリガ位置の4クロック前)でアドレスフェ

〔図8〕PCI-Xのバストレースデータ表示

(a) Clockモード

(b) Transferモード

ーズがあり，メモリリードブロック(MRdBlk)のトランザクションがスタートしました．

(2) Sample「－3」ではアトリビュートフェーズ(Attrib)で，バイトカウント(ByteCnt)は018になっているので，リードするデータは18hバイトであることがわかります．またリクエスタID(ReqID)やタグ(Tag)によりリクエスタの情報を表示しています．

(3) Sample「－2」ではターゲットレスポンスサイクル(Tresp)になっています．

(4) Sample「－1」はコンプリータ側がウェイト状態になっています．

(5) Sample「TRIG」でトリガがかかった位置です．スプリットレスポンス(SpResp)が返されているのがわかります．つまり，コンプリータがすぐに要求されたデータを用意できないため，スプリットレスポンスを返したわけです．

(6) Sample「10」ではスプリットコンプレッション(SpComp)サイクルが起きています．コンプリータが先ほど要求されたデータを，自身がリクエスタとなりデータ要求元に転送(ライト)します．リクエスタIDやタグナンバを確認すると，同一のものであることがわかります．

(7) Sample「11」ではアトリビュートフェーズが入り，バイトカウントは先ほどと同じ18hバイトとなっています．

(8) Sample「13」以降では，コンプリータからリクエスタに対してメモリライトが実行されています．バイトカウントも4バイトずつ減っているのが確認できます．
(9) Sample「18」でこのトランザクションが終了したので，残りのバイト数は000となっています．

● Transferモード

次は，Transferモード(アトリビュートフェーズとデータフェーズでのみトレース，アイドルやウェイトサイクルはスキップ)でトレースして，スタンダードブロックサイズムーブメントのしくみを確認します．このプロトコルは，128バイト境界においてのみトランザクションをディスコネクトすることができます〔**図8(b)**〕．

(1) Sample「TRIG」でメモリライトブロック(MWrBlk)のトランザクションが実行されていますが，ステータス(Status)がデータから次のステージでアローワブルディスコネクトバウンダリ(DNxtADB)が起こることを伝えています．このときの下位アドレスを確認するとF0hとなっています．つまり，この3クロック後にアローワブルディスコネクトバウンダリを実施できます．
(2) Sample「3」で，このトランザクションはディスコネクトされました．
(3) Sample「4」で，新しいトランザクションが始まりました．「3」と「4」のステップ間のタイム差は60nsなので，新しいトランザクションが始まるまで4クロック分バスが開放されていたことがわかります．バイトカウント(ByteCnt)を確認すると010となっているので，残りのバイト数は10hバイトです．
(4) Sample「5」で，新しいトランザクションでデータ転送が再開されています．
(5) Sample「8」で，このトランザクションが終了しています．

このように，バスアナライザを利用すると，各種信号をトレースして，クロックごとの信号状況やステータス，そしてバイトカウントやリクエスタID/タグなど，PCI-Xで必要な情報が確実に確認でき，デバッグをする際にはたいへん強力なツールとなります．

まとめ

PCIバスに替わる標準バスとして，PCI-X以外にも1Gバイト/秒以上のデータ転送レートをもつシリアルバスが注目されています．これらは各デバイスをポイントツーポイントで結び，PCIやPCI-Xのように，多数のデバイスを同じ信号線を共有するインターフェースとは異なります．

今後は，用途に合わせてバスを選択することが必要になってくるかと思いますが，PCIの特徴である「便利で簡単」にさらに「速い」を付加したPCI-Xは，既存のシステムやボード類の資産をそのまま流用でき，アーキテクチャも流用可能なので，シリアルバスに対しても大きなアドバンテージがあります．

現に，ハイエンドサーバなどでは，現時点ですでにPCI-Xが使われてきています．PCI-Xの開発支援ツールも，バスアナライザがリリースされており，デバッグや評価の上で，非常に強力にバックアップされています．

本稿が，今後PCI-Xを検討される際の一助になれば幸いです．

むらい・やすひで (株)ミッシュインターナショナル

Column

66MHz PCIデバイスより，66MHz PCI-Xデバイスの設計が楽なわけ

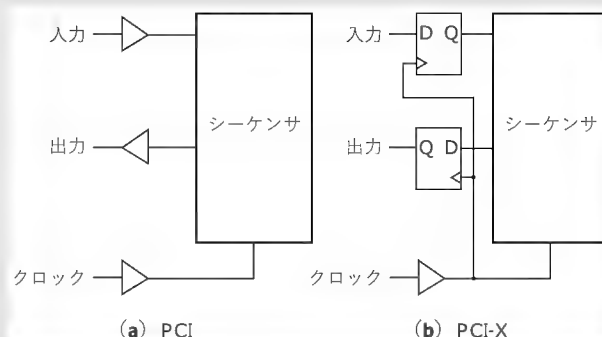
FPGAであろうがASICであろうが，デバイス内部のシーケンサやアドレスデコード条件が複雑になればなるほど伝播遅延が増えます．また，出力ピンから実際に信号が出力されるまでにも出力遅延があります．そして，バスの規格としてのセットアップやホールドタイムの規定があります．

PCIバスは同期バスなので，入力信号をそのまま内部のシーケンサなどで使うことができます〔**図A(a)**〕．ここであるデバイスの伝播遅延が10ns，出力遅延が10nsとしましょう．クロック33MHzのPCIバスではセットアップが7nsと規定されているので，3nsの余裕があることがわかります．しかし，クロックが66MHzになるとどうでしょう．伝播遅延と出力遅延だけで20nsになり，このデバイスは66MHzでは動作しません．

PCI-Xでは，すべての入出力信号をいったんフリップフロップで受けてから出力するように設計します〔**図A(b)**〕．このような構造であれば伝播遅延が出力信号に加算されないので，出力遅延の10nsだけとなり，3nsのセットアップタイムを考慮しても2nsの余裕が残ります．

井倉将実 来栖川電工(有)

〔図A〕PCIおよびPCI-X対応デバイスの入出力部

第6章 PC周辺機器をより高速に接続するための

USBハイスピード伝送の実現

桑野雅彦

USBは1.5Mbpsのロースピードや12Mbpsのフルスピードから，一気に480Mbpsのハイスピードまで高速化させるために，ロースピードやフルスピードで2V程度あった信号振幅を，ハイスピードでは400mVと大幅に小振幅化している．また互換性を重視し，従来のUSBポートに接続しても，フルスピードデバイスとして動作させなければならないため，デバイス内部にはフルスピード用の回路の実装も必要になる．ここではUSB2.0対応デバイスのトランシーバ回路やその動作を解説する．(編集部)

USB2.0ではUSB1.1のロースピード(1.5Mbps)，フルスピード(12Mbps)に加えて，480Mbpsのハイスピードモードがサポートされました．もともとUSBは従来のCOMポートやプリンタ，マウス，キーボードなどを代替えするものとして，安価に中低速のデータ転送を行うことを目的として規格化されました．しかし，その後USBの急速な普及にともない，外部ストレージやビデオキャプチャ関連などへの展開が図られはじめたあたりから，12Mbpsという速度に不満が出てきました．さらにIEEE1394との速度競争などもあり，USB2.0では最高伝送速度が一気に480Mbpsまで引き上げられました．

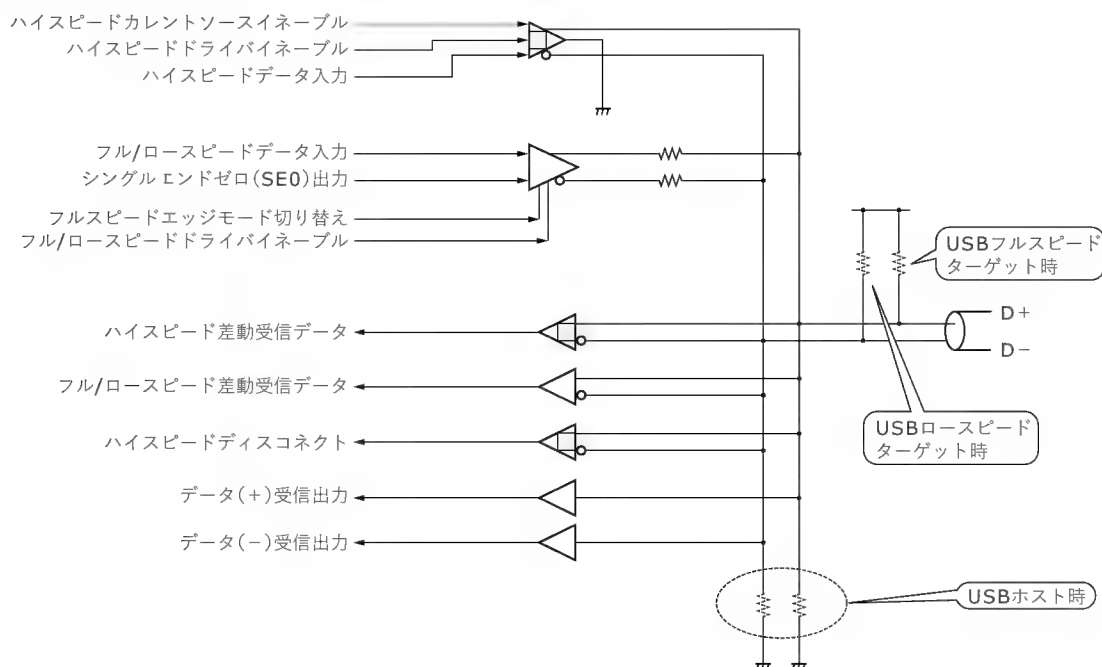
この段階で，従来との互換性維持のためにUSB1.1で使用されていたコネクタやケーブルなどをそのまま流用することや，下位互換性維持のためハイスピードモードをサポートしている機器では，USB1.1のフルスピードモード，またはロースピードモードもサポートする必要があるなど，デバイス側の設計も複雑なものになっています．

● ハイスピードモードでの大きな変更点

USB2.0では，480Mbpsのハイスピードモードの導入にあたり，信号レベルやデバイスの検出方法からプロトコルにいたるまで大幅な見直しが行われました．これらのうち，今回説明する高速伝送に関わるものでとくに大きなものは，次の五つです．

(1) トランシーバの内部構成の変更

(2) 信号レベルの変更(USB1.1では3V系．ハイスピードモードは400mV)

〔図1〕ハイスピードトランシーバ回路

(3) 信号の終端の導入
(4) 浮遊容量など基板設計上の規定が厳しくなった
(5) 信号品質に関してアイパターンによる定義がなされた

● ハイスピードトランシーバ

ハイスピードモードのデバイスは，USB1.x ポートに接続したときにはフルスピードデバイス(12Mbps)で動作することが要求されています．このため，ハイスピードモード対応のトランシーバの内部構造は**図1**(前頁)のような複雑なものとなっています．

ハイスピードデバイスはホストと接続されると，まずハイスピード対応可能であるかどうかのネゴシエーションをホスト(あるいはハブ)との間で行い，ハブがハイスピード対応可能という応答をすればハイスピードトランシーバを生かし，非対応であればフルスピードモードのトランシーバを生かすという動作になります．この切り替えは通常USBコントローラチップ自身で行われますが，ディスクリプタ情報などハイスピードとフルスピードで切り替えが必要なものもあることから，ハイスピードとフルスピードのどちらで動いているのかということは，CPUで検出できるようになっています．

〔**図2**〕**USBフルスピード/ロースピードの接続関係**

〔**図3**〕**USBハイスピードの接続関係**

● トランシーバの接続関係

フルスピード/ロースピードデバイスの送受信用トランシーバ部分を切り出したのが**図2**です．フルスピード/ロースピードデバイスでは信号レベルは3.3V系のロジックとほぼ同じで，デバイスの着脱検出用として，ダウンストリーム側(USBホスト：PCなど)では15KΩの抵抗でプルダウンして，アップストリーム側(USBターゲット側)ではフルスピードならD＋を，ロースピードならD－を1.5KΩでプルアップしています．

一方，ハイスピード時の接続関係は**図3**のようになっています．一見それほど大きな違いはないようですが，抵抗の付きかたと値に注目してください．

ハイスピードモードでは，両端がいずれも45Ωという比較的小さな値の抵抗でグラウンドと接続されています．これは終端抵抗と呼ばれているもので，後で説明するとおり高速伝送のためのキーとなっています．終端抵抗は信号が伝送路であるケーブルを通ってトランシーバICに入る直前に取り付けられるもので，USBの場合同じ電線を双方向で利用するので，両端に終端抵抗が取り付けられています．

● 信号レベルの低減

図3からわかるとおり，USB2.0では45Ωの抵抗が二つ並列接続されるので，信号線とグラウンドの間の抵抗は22.5Ωということになります．もし，USB1.1の時のように“H”レベルが2V程度必要ということになると，一本あたり89mA(2 ÷ 22.5)，最大178mAもの電流が流れることになってしまいます．消費電流の大きさもさることながら，これだけの電流を480Mbpsという高速なレートでスイッチングするというのはかなり難しいことになってきます．また，大きな電流がON/OFFを繰り返すために不要輻射(不必要な電磁波の放射)も増えてしまいます．このようすを**図4**に簡単に示します．高速な電流を高速度でON/OFFするためには信号の急峻な変化を要求されます．これによって高い周波数の不要輻射も増えてしまうわけです．

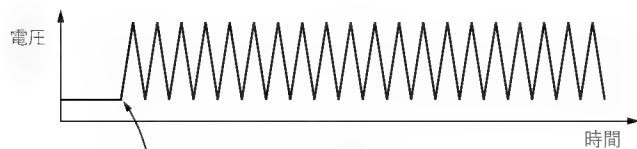
これらを解決するには，信号レベルの引き下げが有効です．**図4(c)**にあるように信号レベルを小さくしていけば，その分信号の急峻さもなくなりますし，流れる電流も小さくなり，消費電流や不要輻射などの防止にも効果的です．ハイスピードモードでは信号レベルを400mVまで引き下げました．これによって信号線に流れる電流は約17.78mA(0.4 ÷ 22.5)まで小さくなっています．

● ケーブルと分布定数回路

ハイスピードモードでは終端抵抗が大事な役割を果たすことになるのですが，それではこの45Ωという値はどうして決まったのか，そもそもなぜ終端抵抗というものが必要なのかということについて，簡単に説明しておきましょう．

まず話を簡単にするため，**図5**に信号線が1本だけの場合の

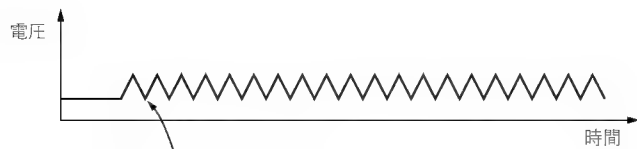
〔図4〕信号レベルを下げる効果

(a) ロースピード/フルスピードの電圧レベルとスイッチング

(b) ロースピード/フルスピードの電圧レベルのまま高速化

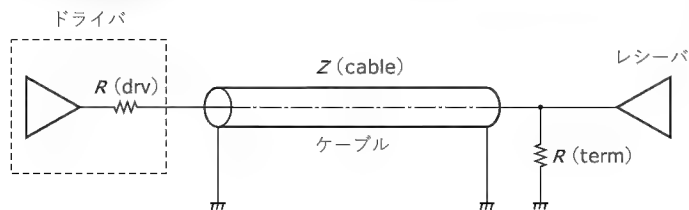
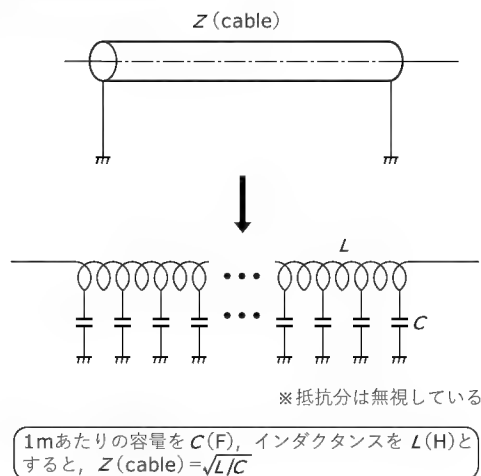
(c) 電圧レベルを下げて高速化

〔図5〕伝送線路のモデル

〔図6〕LCの分布定数回路

〔図7〕信号の反射

伝送路のモデルを描いてみました．ドライバから出た信号がケーブルを通ってレシーバまで接続された状態を示しています．ここで，Zは後で説明するケーブルの特性インピーダンス，R(終端)が終端抵抗の抵抗値を示しています．

さて，ここでケーブル部分を切り出してみます．ケーブルは1本の信号線とグラウンド(シールド)で構成されています．信号線には当然何らかのインダクタンス(誘導性：コイル：成分)が含まれます．また，グラウンドと平行しているので，ここには何らかのキャパシタンス(容量：コンデンサ：成分)が存在します．ケーブルを非常に細かく切り刻んでみれば，信号線に直列にコイルが，グラウンドとの間にコンデンサが入っているものが無数に並んだような状態になっているとみなすことができるわけです．これを回路図のように描いたのが**図6**です．このように非常にミクロなデバイスが無数に並んだような状態としてモデル化される電子回路を，分布定数回路と呼びます．

分布定数回路のインピーダンスを計算するのはなかなか面倒ですが，ケーブルの電気的特性が均一に作られていて，単位長さあたりのインダクタンス(L)やキャパシタンス(C)が一定であり，またケーブルの直流抵抗分が無視できる場合には比較的簡単に整理できます．結果は**図6**にも示したとおり，$\sqrt{L/C}$ であらわされます．これがケーブルの特性インピーダンスで，この値をここではZ(cable)と表しておくことにします．

● 信号の反射

ここで**図7**を見てください．いまZ(cable)の特性インピーダンスをもったケーブルを信号が伝わってきたとします．最終値に任意のインピーダンスZ(term)をもった素子がつながってい

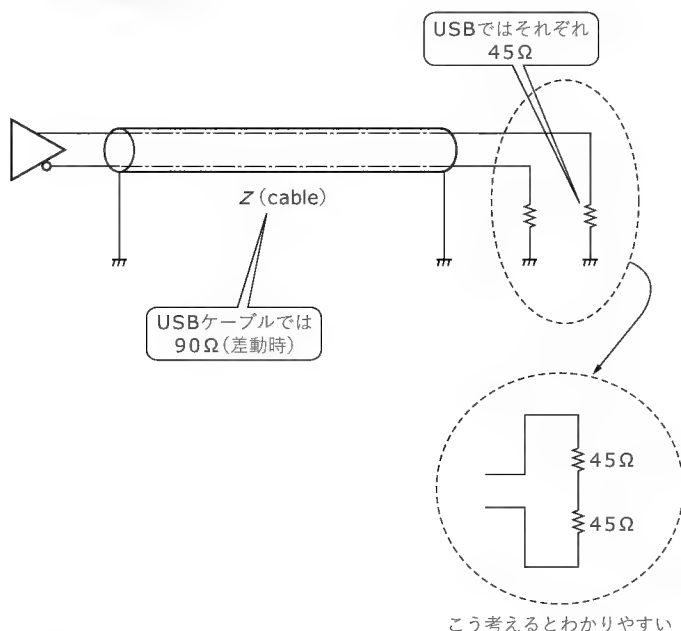
た場合，Z(cable)の値とZ(term)の値が等しくないと，到達した信号のエネルギの一部が終端Z(term)で消費されますが，残りはケーブルを伝って戻っていってしまいます．これを信号の反射と呼び，戻っていく信号を反射波と呼んでいます．コップに波を立てると発生した波がコップに当たって跳ね返るのと同じように，段々小さくなりながら行きつ戻りつするわけです．

単発のパルスだけ伝えるような場合には，最初に信号が到達したことだけ分かれば良いようなものですが，実際には変化する信号で情報を伝えているので，この戻ってきた信号が本来の信号と合成されてしまうと，誤動作を引き起こす原因となります．反射してきた信号はそれほど長い時間往復するわけではないので，伝送速度が充分に低速であるうちはそのような早い変化を取り込まないようにすることで対処可能ですが，伝送速度があがってくると，信号と反射波の区別が難しくなっていきます．

Z(cable)とZ(term)が同一である場合には，理論上反射は発生しません．このような状態を「整合が取れた状態」，負荷の値Z(term)を整合負荷，無反射負荷などと呼んでいます．整合をとることを考えて設計することで，(現実には素子の誤差などもあって完璧に一致させることは不可能なので，わずかながら反射は発生するが)実用上問題ないレベルにまで抑えることが可能となるわけです．

この手のケーブルのインピーダンスは一般に数十Ωから100Ω程度というのが一般的ですので，終端も同じ程度の値の抵抗を使用します．

● USBバスと終端抵抗

USBバスの場合にはデータ線が2本(D＋とD－)あるので少々ようすが変わってきます．USBでは**図8**のようになっています．この場合，終端抵抗の値はケーブルの特性インピーダンスの半分の値にしておきます．USBケーブルではインピーダンスは90Ωと規定されていたので，終端抵抗は45Ωとなります．これが，すでに説明したUSBの終端抵抗値45Ωの根拠です．

USBケーブルの特性インピーダンスはUSB1.1の時代からすでに90Ωと決められていました．ところが，USB1.1の12Mbps程度の伝送速度の場合，かなりいいかげんな値にしていても問題なく伝送できてしまうことから，一部に粗悪なケーブルが流通しており，ハイスピードモードが規格化された当初，この点が問題として掲げられていました．ケーブルのインピーダンスが違っていれば，終端抵抗と不整合になってしまい，反射による影響が出てしまうわけです．

終端抵抗の値が決まり，信号レベルを決めれば流れる電流も決まってきます．信号レベルを引き下げるほど不要輻射や消費電流などの面でも有利にはなりますが，一方で外部からのノイズの影響など，外乱を受けやすくなってきますので，むやみに下げると問題が出てきます．このトレードオフでどの程度の信号レベルにするかが決まってくるわけですが，USB2.0のハイスピードモードでは400mVということにしています．

ちなみに同様に高速なデータ伝送が必要になっているパソコン用CPUやチップセット間の接続信号も昔はTTLレベルが一般的でしたが，現在では新製品が出るたびに電圧が引き下げられる方向になっています．

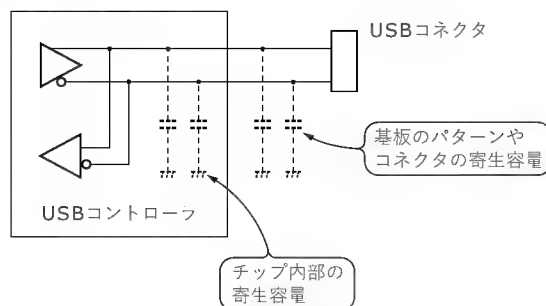
● 寄生容量と信号の立ち上がり/立ち下がり

部品としてのコンデンサを取り付けなくても，2本の信号線や信号線とグラウンドの間には必ず何らかのキャパシタンス(容量)が存在します．これを寄生容量と呼びます．**図9**にチップやパターンの寄生容量の存在を図にしてみました．

コンデンサの場合，容量をC(F)とすると，ある周波数f(Hz)におけるインピーダンスは$1/(2 \times \pi \times f \times C)$となります．それほど高速ではない信号を扱うときにはそれほど大きな問題とならない寄生容量も，扱う信号の周波数が大きくなると無視できなくなってきます．

コンデンサのインピーダンスが低くなるということは，コンデンサを通る電流が増えるとういことであり，電流が増えると

〔図8〕差動伝送の場合

〔図9〕寄生容量

いうことは信号として伝わっていく分が減る，すなわち減衰してしまうということになるわけです．

高い周波数成分ほど減衰することから，信号波形も急峻さがなくなり，なまったようなものになってきます．こうなると高速な伝送にとって困ったことになります．

逆に信号波形が急峻すぎる場合には，高い周波数成分も多く，流れている電流も多くなっているので，不要輻射などの発生が懸念されます．とくにロースピード（1.5Mbps）の場合，ケーブルのシールドはしなくてもよいことになっているので，不必要に急峻な波形にすることはできません．

このようなトレードオフの中で，USBでは，

(1) ロースピードデバイス
- USBデバイスのドライバ負荷200pF～600pFの容量負荷が可
- 負荷接続時の立ち上がり/下がりは75ns～300ns

(2) フルスピードデバイス
- ドライバ負荷は50pFまで
- 立ち上がり/立ち下がりは4～20ns

という規定になっています．さらにハイスピードデバイスの場合には立ち上がり/立ち下がりは500psとなりますが，480Mbpsの伝送ともなるとこの程度の規定では不足なため，より厳密なアイパターンによる定義が加えられています．

● アイパターン

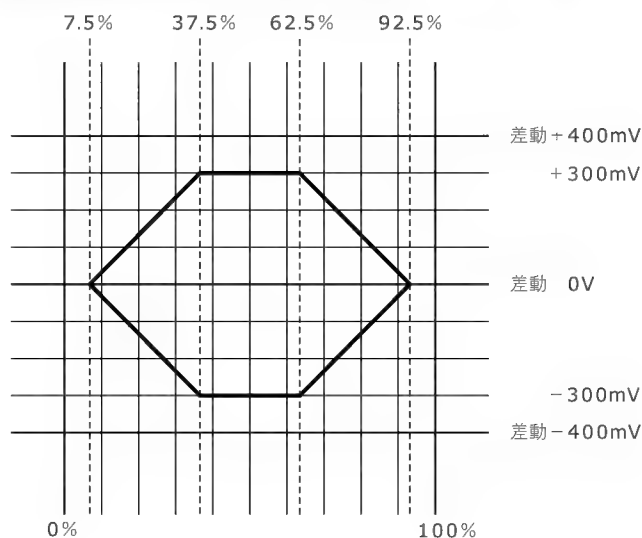
アイパターンについてはページ数の都合もあり詳しく触れられませんが，USBの規格書から一つだけ例を引き出しておきました（**図10**）．これはドライバが無負荷状態での出力コネクタ部分での規定です．見方は，上下方向が電圧レベルで，左右方向が時間です．電圧が0Vを中心にしているのは差動電圧でみているためです．D＋とD－の電圧が同じときが0V，D＋が＋400mV，D－が0Vなら＋400mV，逆ならば－400mVということになります．横軸は時間で，0%から100%までが1ビット分の時間です．

図の中央部分に描かれた算盤の玉のような六角形がちょうど「目」のように見えるので，アイパターンと呼ばれます．信号の変化がこの目の内側を通らないようにしなくてはならないというのがその規定です．

イメージとしてはオシロスコープで伝送波形を見ている状態です．高速に伝送していると，伝送波形がひっきりなしに動いていることになりますが，ディジタル信号である以上，‘1’か‘0’かが確実に確定している時間があるため，輝線で真っ白にはならず，ぽっかりと穴があいたような場所ができます．アイパターンはこの穴の形状が最小でもこれだけなくてはいけないという規定に相当します．

また実際の波形写真としては，次号以降に掲載予定の『USB compliance Testの概要』が参考になるかと思います．

USBの仕様書ではさらに細かく場合分けされたアイパターンが指定されており，480Mbpsの高速伝送を確実に行うためのガイドラインとなっています．

〔図10〕USBハイスピードのアイパターン例

● ケーブルスキュー

USBは二本のデータ線（D＋とD－）を利用し，通常は片方が“H”レベルならもう一方は“L”レベルにするという差動伝送を行っています（ただし，リセットなど特殊な条件を作るため両方“L”や両方“H”という状態になることもある）．レシーバはこの両方のデータ線の電位差を見て“1”か“0”かを判定しているので，もし，このときD＋とD－の到達時間に差ができてしまうと，データを誤読する可能性がでてきます．

この両者の到達時間差をスキューと呼びます．USBケーブルの場合，データ線のスキューは100ps以下になるよう規定されています．

＊　＊

USB2.0ではハイスピード対応のため，この他にも細々した規定が付け加えられています．あくまでも下位互換性を維持しながらハイスピード伝送に対応するためにさまざまな工夫がなされているので，興味ある方はさらに細かい部分まで調べてみると面白いでしょう．

くわの・まさひこ　パステルマジック

Appendix

IEEE1394.b最新動向

北山洋幸

はじめに

現在ではIEEE1394.a-2000に準拠したインターフェースが，多くのパソコン，ディジタル家電，ハードディスク，およびFAネットワーク機器などに搭載されています．USB2.0が出現し，IEEE1394.a-2000の速度面での優位性はなくなりました．とはいえ，USB2.0はやっと出回ったところです．ノートパソコンでは，やっとUSB1.1からUSB2.0へ移行が始まったばかりです．このような状況や使用目的の違いから，IEEE1394とUSBは併用されています．

IEEE1394はIEEE1394.bの出現により，やっと普及が始まったUSB2.0と比較して，速度や距離で再び優位に立ちました．すでにIEEE1394.b対応の800Mbpsの製品が出荷され始めました．現在は800Mbpsの製品が出始めたばかりですが，IEEE1394.bは最高3.2Mbpsまで規格されています．近い将来には，より高速な製品が現れることでしょう．もっともIEEE1394.bが発表されてから，すぐに製品が現れたわけではありません．最近になって，やっとLSIやそれを採用したHDDなどが現れ始めました．

IEEE1394.bにはネイティブのベータ(beta)モードと，既存のIEEE1394-1995，およびIEEE1394.a-2000と互換性がある，バイリンガル(bilingual)モードの二つが用意されます．速度も新たに800Mbps，1.6Gbps，3.2Gbpsが追加されました．不評だった距離の問題も解決されています．しかし，ケーブルやコネクタの種類は増え，既存のIEEE1394で使われていたコネクタやケーブルとの組み合わせを考えると，組み合わせは少々複雑になりました．

● 従来までのIEEE1394

IEEE1394の最初の技術仕様が完成したのは1987年頃です．この仕様は1995年に正式にIEEE Std 1394-1995として認められました．2000年には，IEEE1394インターフェース規格の補完・拡張規格であるIEEE1394.aがIEEE Std 1394.a-2000として採択されました．追加されたおもな特徴としては，4ピンコネクタやバスリセット動作の安定化，ショートバスリセット(バスリセットの時間を短くする機能)，物理層の低消費電力化(サスペンド状態などの追加)などが挙げられます(**表1**)．

USB2.0の出現により，IEEE1394.a-2000の速度面の優位性はなくなったと考える人もいますが，USB2.0はあくまでもパソコンをおもなマーケットとしており，IEEE1394と用途が異なります．このためIEEE1394とUSBは共存していくと思われます．USBでもピアツーピア接続に対応できるように，USB2.0の補完規格としてUSB On-The-Go(OTG)が策定されましたが，OTG対応を正式にうたった製品の登場はこれからという状況のようです．

● IEEE1394.bで改善されたおもな点

IEEE1394-1995からIEEE1394.a-2000への変更はそれほど大きなものではなく，問題を引き起こしていたバスリセット期間の短縮や4ピンコネクタの追加でした．それに比較して，IEEE1394.bでは大きな変更がいくつもあります．まず，従来のDSポート(DS/LINK)にβポート(ベータポート，8B10B符号化)が追加されました．メディアもたくさん現れ，それと同時にコネクタ形状も複数現れました．性能(とくにスループット)を向上させるために，アービトレーションもデータ転送と平行して行えるようになりました．また不満の多かった距離の問題も解決されました(**表2**)．

高速化は単純なスピードの高速化だけでなく，スループットも考慮されています．従来のIEEE1394で課題だった，ギャップによるアービトレーションでは，速度をいくら速くしてもバスを効率的に使用することはできません．そこでアービトレーションの方法を根本的に変更しました．また，転送速度の高速化に伴い，符号化を従来のDS/LINKから実績のある8B10Bへ変更しました．

● アービトレーション

IEEE1394.bでは，既存のIEEE1394と違い，アービトレーションをデータの送信と平行して行えるようになりました．今までのIEEE1394ではデータの送受信が終わった後に，アービトレーションを行っていたためバスを有効利用できない時間帯がありました．しかし，IEEE1394.bでは，データの送受信を行っている最中にアービトレーションを行います．このため，データの送受信が終わって，すぐに次のデータの送受信を行うことができます．これによりバスの使用効率が高くなります．

● 符号化の変更

IEEE1394.bでは，データの符号化にGigabit EthernetやFibre Channelで使われている8B10Bを使うようになりました．既存の1394で使われていたDS/LINKは使用していません．8B10Bについては，第4章のコラム2(p.89)を参照してください．

一般的に，上記のアービトレーションと符号化を採用したモードを「ベータモード」と呼び，従来のモードをレガシーモードと呼びます．

IEEE1394.bでは，スピードと距離の関係が，今までのように単純ではなくなりました．そこで表に距離，速度，およびメディアのマトリックスを**表3**に示します．

〔表1〕IEEE1394-1995に対するIEEE1394.a-2000の特徴

●4ピンコネクタの追加
●バスリセット動作の安定化
●ショートバスリセット(バスリセットの時間を短くする機能)の追加
●PHYの低消費電力化(サスペンド状態などの追加)

〔表2〕IEEE1394.bで追加されるおもな特徴

●高速化(800Mbps～3.2Gbps)
●長距離化(～100m)
●伝送媒体の追加(石英ファイバ，プラスチックファイバ，UTPなど)
●光ファイバのための仕様の追加(初期化の方式，符号方式など)

● 互換性

IEEE1394.bでは転送の符号化も違うし，コネクタの形状も異なるため，これまでのポートにそのままで接続することはできません．800MbpsのIEEE1394.bでは9ピンのコネクタが使われます．この9ピンのコネクタと，従来からある4ピン，6ピンを9ピンコネクタに接続するためのケーブルが用意されます．また，IEEE1394.bでは純粋なIEEE1394.bの規格であるベータモードと，下位互換性を保証するためのレガシーモードが用意されます．

● 製品動向

USB2.0はすでに一般的なものになりつつあります，それに比べIEEE1394.bはまだまだ普及しているとはいいがたい状況です．

IEEE1394.b対応のパソコンとしては，アップルの新G4 Macがあげられます．800MbpsのIEEE1394.b（アップルではFireWire800と表現）と同時に400MbpsのIEEE1394.aとUSB2.0も搭載しています．

Macworld Conference&Expo/San Francisco 2003（2003年1月7日～10日）で，いくつかのベンダから800Mbps（IEEE1394.b）の対応の外付けHDDが発表されています．これらは，ほとんどが，Oxford Semiconductor社（http://www.oxsemi.com/）のIEEE1394.bコントロールチップOXUF922を搭載したインターフェースボードを採用しているようです．OXUF922はIEEE1394.bだけでなくUSB2.0にも対応しています．このため発表されたほとんどのHDDがIEEE1394.bとUSB2.0を搭載しています．Oxford Semiconductor社はIDEブリッジで有名な会社です．OXUF922以外にも，いくつものブリッジLSIや評価ボードを販売しています．

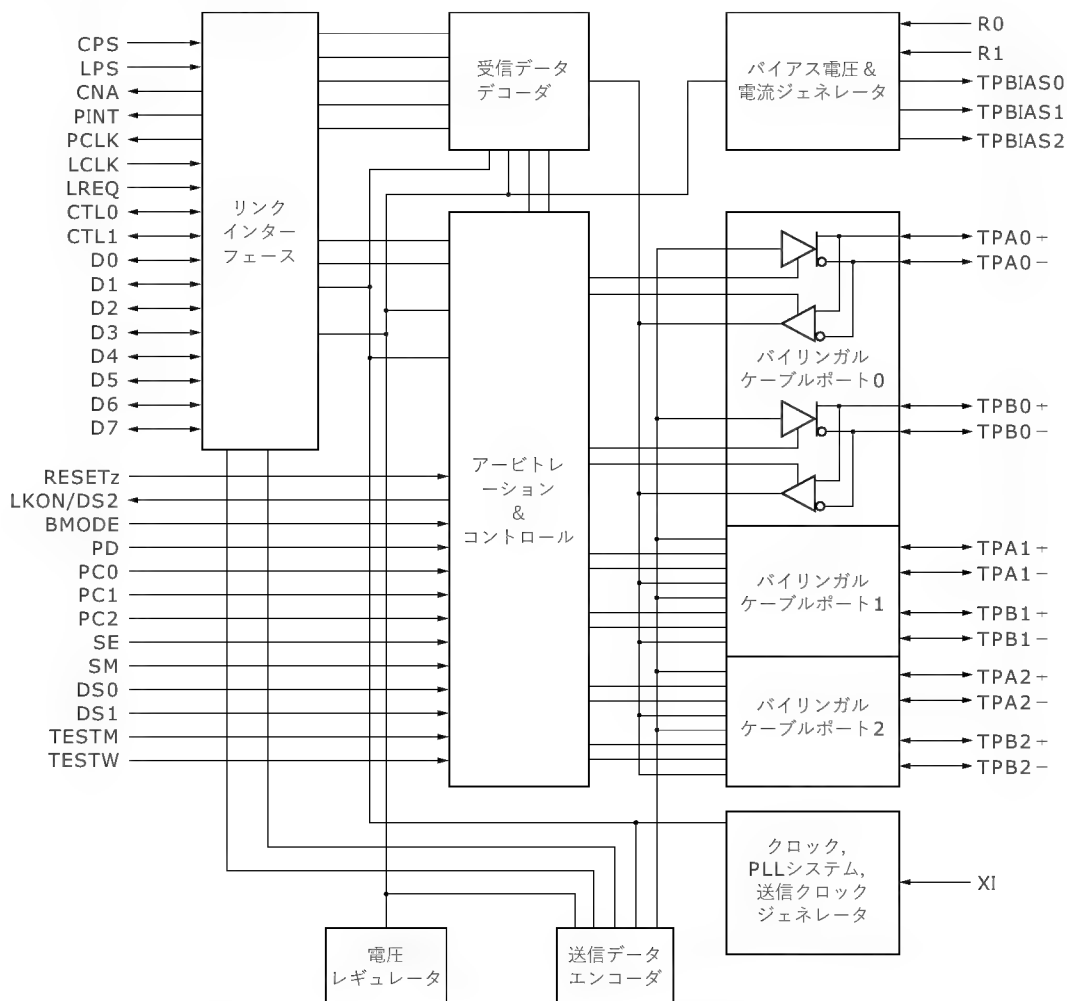
HDDに関しては日本でも，ロジテックからFireWire 800に対応したモデルが発表されています（**写真1**）．

ほかにも数社が64ビットPCIを用いたIEEE1394.bボードの販売を進めているようですが，本原稿執筆時点では，「玄人志向」のIEEE

〔表3〕各種メディアと距離，速度

メディア	100Mbps	200Mbps	400Mbps	800Mbps	1600Mbps	3200Mbps
STP	4.5m	4.5m	4.5m	4.5m	4.5m	4.5m
UTP	100m	－	－	－	－	－
POF	50m	50m	－	－	－	－
HPCF	100m	100m	－	－	－	－
GOF	100m	100m	100m	100m	100m	100m

STP ：9ピンシールドツインスペアケーブル
UTP5 ：CAT-5 UTP（ISO/IEC 11801 ch.7）
POF ：Step-indexプラスチック光ファイバ
HPCF ：Hard Polymer Clad光ファイバ
GOF ：ガラス光ファイバ

〔図1〕IEEE1394.b対応PHYデバイスTSB81BA3

〔写真1〕IEEE1394.b に対応した HDD（ロジテック製）

（a）外観

（b）IEEE1394.b 対応 9 ピンコネクタ

1394B-PCI64 が現れているのみです（今後も安定供給が続くか不明だが）．この PCI ボードは，LINK に Indigita 社（http://www.indigita.com/）の iND60C20 が，PHY に TI 社の TSB81BA3A が使われています．64/32 ビット PCI に対応し，9 ピンのバイリンガルを 3 ポート装備しています．

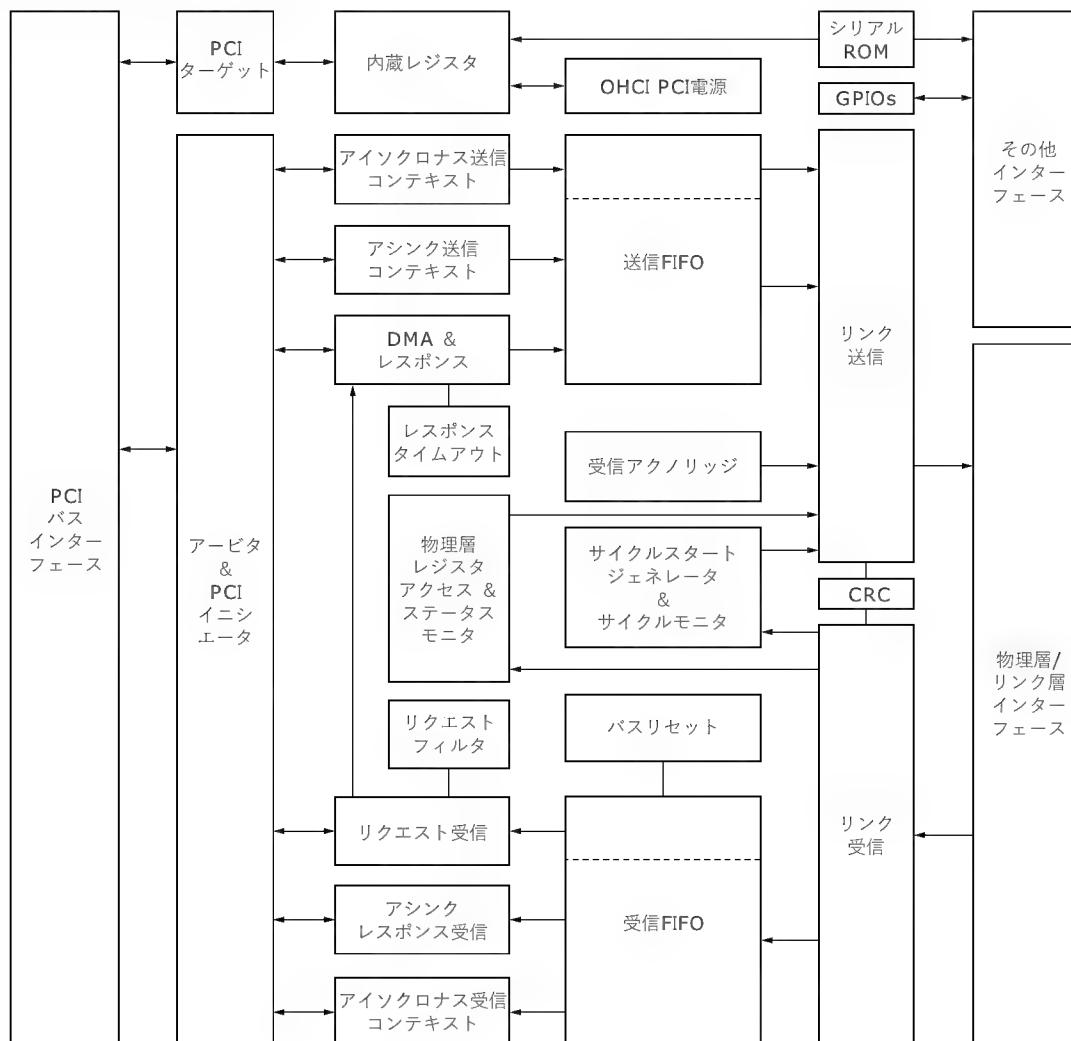
● LSI

IEEE1394.b 対応の LSI もやっと入手可能になりました．先に述べた，Oxford Semiconductor 社の LSI もありますが，このチップは IDE ブリッジとして設計されています．よって IEEE1394.b のアプリケーションを開発するには向きません．

テキサス・インスツルメンツから IEEE1394.b のリンク層（LINK）チップとして TSB82AA2 が，物理層（PHY）チップとして TSB81BA3 が販売されており，これが現時点で使える IEEE1394.b 対応 LSI といえるでしょう．

TSB81BA3 は，3 ポートのバイリンガル PHY デバイスです（**図1**，前頁）．最高 800Mbps に対応した，IEEE1394.b 標準規格準拠製品です．この TSB81BA3 は前世代の IEEE1394-1995 や IEEE1394.a-2000 の 2 倍の速度を実現するだけでなく，伝送距離を最高 100m まで延長しています．また，従来の IEEE1394.a-2000 への下位互換性を備え，DS/LINK 符号化方式，ならびに IEEE1394.b の 8B/10B 符号化方式をサポートします．

この PHY と対で使用する LINK デバイスが TSB82AA2 です（**図2**）．TSB82AA2 は，OHCI（オープンホストコントローラインターフェース）1.1 規格に準拠するとともに，OHCI1.2 にも対応します．OHCI

〔図2〕IEEE1394.b 対応 LINK デバイス TSB82AA2

はパソコンだけでなく，組み込み機器でも広く使われるようになりました．このOHCI規格は，より高速のアシンクロナスおよびアイソクロナスデータ伝送，アドバンスドパワーマネジメント機能をサポートしています．

● IEEE1394.bではPCIバスがネック

IEEE1394.b対応の周辺機器が発表されないのは，単純にLSIの供給遅れだけではない問題があります．現在のパソコンではすでにかなり前からCPUはGHz帯に入っています．CPUやビデオまわりは，ずいぶんと高速化が図られました．しかし，I/OとなるPCIインタフェースは相変わらず32ビット/33MHzが使われています．しかも，このバスは汎用なので，一つのデバイスが占有して全バンド幅を使うわけではありません．そう考えると，IEEE1394.bで追加されたもっとも遅い速度である800Mbpsさえ，PCIバスがボトルネックとなる可能性が高いのです．つまり，現在のパソコンアーキテクチャではIEEE1394.bの性能を使い切ることは簡単なことではありません．現在のパソコンであればIEEE1394.aで十分な性能を提供していると言えるでしょう．

同じような問題がシリアルATAやGigabit Ethernet[注1]にも存在します．これらはチップセットへ組み込まれるのではないかと言われています．ところがIEEE1394.bに関しては，パソコンへどのように搭載されるのか明確ではありません．このあたりがIEEE1394.b製品が現れない一因でしょう．

筆者の会社においても，すでにIEEE1394.bの開発キットのプロトタイプは完成していますが，いまひとつ本格的な販売に移れない要因がここにあります．もっとも簡単な方法は64ビットPCIを使う方法ですが，これではマザーボードが限定されます．性能は発揮できなくても32ビット作った方が使いやすいのだろうかと思案中です．

いずれにしても，IEEE1394.bの性能をパソコンユーザーが十分に引き出すにはチップセットに入るか，PCI Expressなどを待たなければないのでしょう．800Mbpsで，この状態です，1.6Gbpsや3.2Gbpsでは，小手先の対応では性能はバスネックになり，ほとんど性能を発揮することはできないでしょう．

もっともIEEE1394.bの応用はパソコンだけに限られているわけではありません．FAや組み込み分野では，距離を伸ばしたいユーザーも多いので，しばらくは速度よりそちらに応用される可能性が高いと思われます．実際，車などでIDB-1394（車載用規格）の検討も進んでいます．FAネットワークでも距離に不満が出てたので，そちらがパソコンより先にIEEE1394.bへ興味を示す可能性は高いと予想できます．

きたやま・ひろゆき　（有）スペースソフト
http://www.spacesoft.co.jp/

注1：インテルはGigabit EthernetとのインターフェースとしてCSAをチップセットに組み込むことを発表した．CSA（Communication Streaming Architecture）のバンド幅はPCIの2倍である266Mバイト/秒．CPUとメモリを制御するノースブリッジに直結されるようだ．

Column
IEEE1394.bのコネクタとケーブル

IEEE1394.b専用のコネクタをベータポート，従来の400Mbps以下のスピードの信号も接続できるコネクタをバイリンガルポートと呼びます．**図A**にIEEE1394.bのソケット形状を，**表A**にピン配置を示します．**図A**を見るとわかるように，ベータポートはでっぱりの部分が広く，バイリンガルポートは狭くなっています．よってベータソケットにはバイリンガルコネクタは入りません．逆にバイリンガルソケットにはベータコネクタが入るので，IEEE1394.bネイティブで接続するケーブルは，両端が9ピンのベータコネクタになっています．

またバイリンガルポートには従来の6ピンや4ピンの信号も接続できることになるので，IEEE1394.b対応のケーブルには，次のようなバリエーションができることになります．

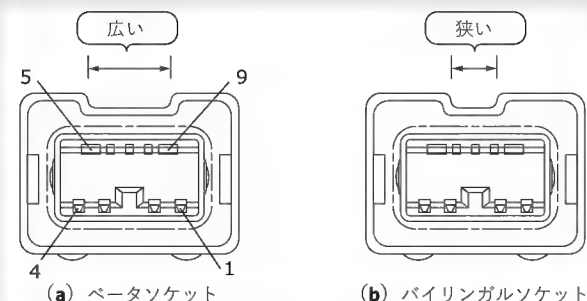
- ベータ対ベータを接続
  →両端バイリンガルコネクタのケーブル
- ベータ対バイリンガルを接続
  →両端バイリンガルコネクタのケーブル
- バイリンガル対バイリンガルを接続
  →両端バイリンガルコネクタのケーブル
- バイリンガル対6ピンを接続
  →片端バイリンガル-片端6ピン
- バイリンガル対4ピンを接続
  →片端バイリンガル-片端4ピン

〔表A〕IEEE1394.b対応9ピンコネクタ配置

ピン番号	名　称
1	TPB −
2	TPB ＋
3	TPA −
4	TPA ＋
5	TPA用ツイストペアシールド
6	GND
7	NC
8	V_{pp}
9	TPB用ツイストペアシールド

〔図A〕IEEE1394.b対応9ピンソケット形状

(a) ベータソケット

(b) バイリンガルソケット

10Gigabit Ethernetの技術動向

松本信幸

Gigabit Ethernet対応のLANカードやハブが安価で入手できるようになってきた．ブロードバンドの普及もあり，ネットワークの高速化の要求は今後もますます高まるだろう．最後の章は，今回取り上げたバス/インターフェースの中でももっとも長い伝送距離を高速に通信することを考えた，Gigabit Ethernetおよび10Gigabit Ethernetの技術動向について解説する．
（編集部）

10Gigabit Ethernetは，IEEE802.3委員会が検討している，Ethernetの流れの上にある高速インターフェースです．ネットワーク上でやり取りされる情報の多様化や，その量の増大に対応するために，初期のEthernetから見ると，動作メカニズムは大幅に変化しています．

● Gigabit Ethernet

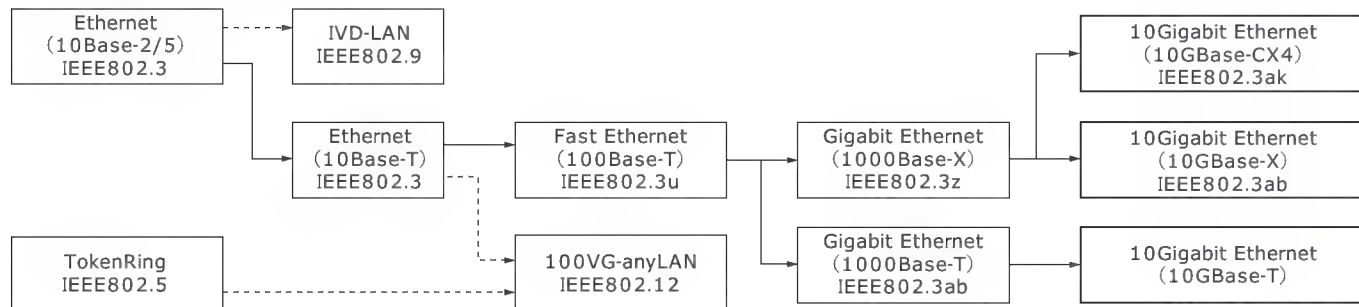
Ethernetの速度は，10倍ずつ速くなっています．おもなものとして（過去には1Mbpsなどというものもあったが）10Mbpsである10Base-T/5があり，次に現在もっとも多く出回っている100Base-TXが続きました．ネットワーク利用者の増加にともない，回線の容量や中継を行う機器のスイッチング能力が増大しましたが，ネットワークの能力の増大はアプリケーションなどの変化を呼び，回線容量のさらなる増加を要求しました．

こうして誕生した技術がGigabit Ethernetです．とはいえ，アプリケーションに変化が出たといっても，回線容量が増加する最大の要因はネットワーク利用者の増加にあるので，Fast Ethernet以下の速度をもつ端末を収容し，基幹部分として利用されることが主目的となります．初期に市場に出た，端末側に10Mbpsインターフェースのスイッチング HUBがもつアップリンク用のインターフェースがFast Ethernetだったようなものです．端末側のインターフェースがFast Ethernetになってしまったため，再びアップリンクの容量が足りなくなり，10倍の速度をもつGigabit Ethernetが必要になりました．ほかにも，その時代の状況に応じていくつか派生していますが，本流は基本に忠実なものとなっていました（**図1**）．

ここまではLANの話なのですが，もう一つの要因としてLANからみて外部へのアクセスが激増したということもあげられます．LANから見て外部へのアクセスは，昔は全体の2割程度で，8割は内部の通信でした．しかし，現在では逆転どころかそのほとんどが外部との通信となっています．もっともこれにはメールサーバなどを社外の事業者に委託しているなどという運用形態の変化も要因の一つです．

こうして高速化していくインターフェースですが，単に動作クロックを早くすればよいというものではありません．信号とノイズの区別がつきにくくなったり，伝送速度によって距離に制限を受けたりという物理的な制約が障壁となって現れてきます．これらに対処するため，Gigabit Ethernetでは目的に応じて複数のインターフェースを用意するようになっています（**表1**）．

これは，一部例外はあるものの，大きく「構成の変更がほとんどない長距離（正確には中距離）伝送用」と，「頻繁に接続変更が可能な至近距離用」に分けられています．前者の検討を行っていたのがIEEE802.3zで，後者がIEEE802.3abです．

〔図1〕10Gigabit Ethernetへの流れ

〔表1〕4種類のGigabit Ethernet

1000Base-CX	スイッチングクローゼットやコンピュータルームなどの室内で用いる
1000Base-T	おもにフロア配線に用いる(アクセス系)
1000Base-SX	低価格光通信用．フロア配線および小規模バックボーンに利用する
1000Base-LX	キャンパスレベルのバックボーンに利用する

〔表2〕Gigabit Ethernetにおける伝送距離

	ファイバ	コア直径	伝送距離
1000Base-SX	MMF	62.5um	275m
1000Base-LX	MMF	62.5um	550m
1000Base-SX	MMF	50um	550m
1000Base-LX	MMF	50um	550m
1000Base-LX	SMF	9um	5km

	ケーブル形状	伝送距離
1000Base-CX	シールドケーブル	25m
1000Base-T	ツイストペアケーブル	100m

〔図2〕CSMA/CD方式

IEEE802.3zで検討されていたインターフェースには，光ファイバケーブルを用いるものと同軸ケーブルを用いるものの2種類がありますが，後者の同軸ケーブルを用いるインターフェースである1000Base-CXはケーブル長が最大で25mと短く，用途も限られているためほとんど市場に出ていません．

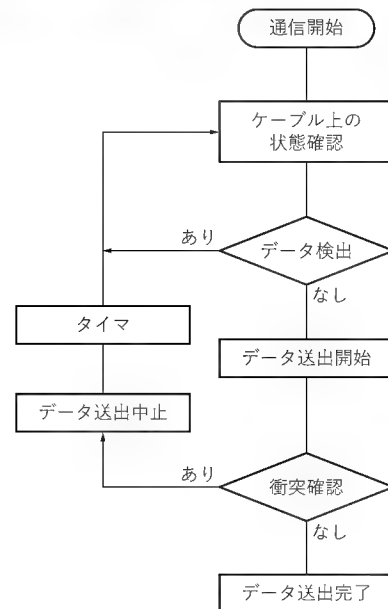
前者の光ファイバケーブルを用いるものには，短波長の光(波長が1000nm未満の短い波長の光：850nm)を用いる1000Base-SXと，長波長の光(波長が1000nm以上の長い波長の光：1310nm)を用いる1000Base-LXがあります．

利用方法の差は，1000Base-LXが長距離用で，1000Base-SXが多少安価な中短距離用となっています(**表2**)．

しかし，スイッチ間の接続であれば光ファイバを用いても大して問題はありませんが，接続する相手がサーバとなると配線はフロア内となりますし，接続の変更(装置の移動などによる)も考えられます．多くの場合，光ファイバケーブルの取り扱いができる人のいない場所で利用されることが考えられるので，レイアウトの変更などが容易に行えるインターフェースが必須であり，IEEE802.3abとして1000Base-Tが検討されました．

これは，従来の10/100Base-TXで利用しているRJ-45コネクタを利用するものです．おもに中継装置間に用いるIEEE802.3zと異なり，端末接続を想定しているIEEE802.3abでは，Ethernetの代名詞ともいえたCSMA/CD(Carrier Sense Multiple Access with Collision Detection)方式も検討されています．しかし，従来と同じ構成では運用面に無理があるので，「キャリアエクステンション」,「フレームバースト」といった対応も選択肢として加わっています．

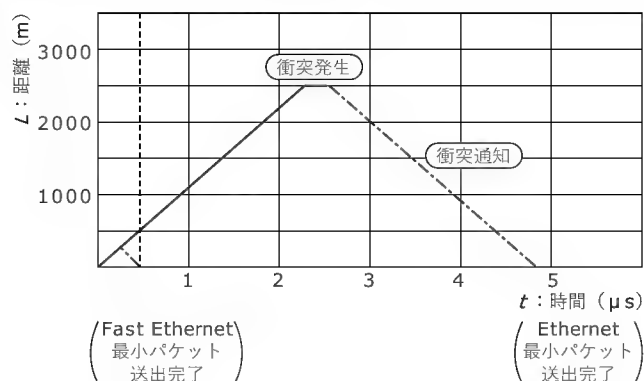
● CSMA/CD方式の限界

Ethernetといえば，もともとは1本の同軸ケーブルに複数の端末を接続し，通信を行うための環境を提供するものであり，ネットワーク上にマスタのいない状況で各々の端末が自立的に通信を実現するものでした．そして，その動作を実現するカギとなる手法がCSMA/CD方式でした．

CSMA/CD方式とは，ネットワークに対してデータを送出する際，ネットワーク上にほかからのデータがないことを確認したうえで送出を開始します．また，データの衝突が発生した場合はそれを検出できるようになっており，そのときはデータの送出を停止し，ある程度の時間待ってから再送を開始します(**図2**)．

ここでインターフェースの速度が高速化していった場合を考えてみると，たとえば速度が10倍になるということは，逆に同じデータ量を送るための時間が10分の1になるということです．電気信号(または光であっても)には速度がある以上，ケーブル内を進むのにも時間を要します．時間が10分の1になるということは，信号が進む距離も10分の1になってしまいます(**図3**)．

つまり，インターフェースの速度が増加するにつれて，CSMA/CD方式の動作できる範囲は小さくなっていきます．先のキャリアエクステンションは，Gigabit Ethernetの25mというCSMA/CD上の距離制限を，200mまで延ばすために用いますが，このように特別に手を打たなければ，従来のままでは利用できなくなっているのが現状ですし，この手もこれ以上の速度の増加には対応できません．

事実上，EthernetにおけるCSMA/CD方式の利用は，IEEE802.3ab以降，対応をあきらめています．しかも，現実問題としてはCSMA/CD方式は伝送路のリソースをむだ遣いするので需要は少なく，IEEE802.3abにおいても，キャリアエクステンションやフレームバーストは「選択肢」でしかなく，現状でもほとんど利用されていません．

〔図3〕コリジョンドメインの変化

● 10Gigabit Ethernet

利用者の増加とアプリケーションの変化は，EthernetをLAN（限定範囲内）のネットワークからインターネットへと変化を促しました．最大の特徴は，通信相手との距離が桁違いに遠くなったということになります．その場合の影響をもっとも受けるのは伝送距離です．ネットワークの利用形態が同じで，アプリケーションにのみ変化がある場合，問題となるのは帯域だけですが，利用形態そのものに変化が出る場合，伝送距離も，より遠くに伝送できるものが必要となります．

10Gigabit Ethernetは，Gigabit EthernetのうちIEEE802.3zをさらに発展させたものとして，IEEE802.3adとして検討されました．Gigabit Ethernetで利用された短波長（1000Base-SX）

〔表3〕10Gigabit Ethernetのインターフェース分類

	10GBase-SR	10GBase-SW	10GBase-LX4	10GBase-LW4	10GBase-LR	10GBase-LW	10GBase-ER	10GBase-EW
インターフェース条件								
伝送路速度（L1）（MHz）	10,312.50	9,953.28	3,125 × 4	2,488.32 × 4	10,312.50	9,953.28	10,312.50	9,953.28
利用波長数	Serial		WWDM（× 4）		Serial			
データ伝送能力（L2）（Mbps）	10,000		10,000		10,000		10,000	
基準波長（λ）（nm）	850		1310				1550	
フレームフォーマット		OC-192c				OC-192c		OC-192c
コーディング	64B66B	64B66B	8B10B		64B66B	64B66B	64B66B	64B66B
Transmitter Type	VCSELs		LASER					
光パワー								
最大光量（dBm）	−4				0	−3		
最小光量（dBm）	−9				−5	−8		
最大受光感度（dBm）					0			
最小受光感度（dBm）	−17				−14			
使用光ファイバ種別（1）	−		SMF					
最大伝送距離（m）	−		10,000				40,000	
使用光ファイバ種別（2）	MMF		−		−		−	
コア/クラッド径	62.5/125		−		−		−	
モーダルバンド幅（MHz×km）	160		−		−		−	
最大伝送距離（m）	28		−		−		−	
使用光ファイバ種別（3）	MMF		−		−		−	
コア/クラッド径	62.5/125		−		−		−	
モーダルバンド幅（MHz×km）	200		−		−		−	
最大伝送距離（m）	35		−		−		−	
使用光ファイバ種別（4）	−		MMF		−		−	
コア/クラッド径	−		62.5/125		−		−	
モーダルバンド幅（MHz×km）	−		500		−		−	
最大伝送距離（m）	−		300		−		−	
使用光ファイバ種別（5）	MMF				−		−	
コア/クラッド径	50/125				−		−	
モーダルバンド幅（MHz×km）	400				−		−	
最大伝送距離（m）	69		240		−		−	
使用光ファイバ種別（6）	MMF				−		−	
コア/クラッド径	50/125				−		−	
モーダルバンド幅（MHz×km）	500				−		−	
最大伝送距離（m）	86		300		−		−	
使用光ファイバ種別（7）	MMF		−		−		−	
コア/クラッド径	50/125		−		−		−	
モーダルバンド幅（MHz×km）	2000（次世代）		−		−		−	
最大伝送距離（m）	300		−		−		−	

と長波長(1000Base-LX)のほかに，1000Base-LXで利用された1310nm帯のものよりさらに波長の長い1550nm帯が追加されています．このインターフェースは長距離伝送に用いるもので，最大40kmまでの伝送を可能としています．

また，1310nm帯では，波長多重であるDWDMを用いたものも加わっています．1310nm帯のDWDMを含めて4種類の波長を用いるようになっているうえに，その各々において2種類のフレームを定義しているので，大枠8種類のインターフェースから目的に応じて使い分けられるようになっています(**表3**)．

● インターフェース別の分類

10Gigabit Ethernetにおけるインターフェース種別の表現も，従来と同じような表記方法となっています(**図4**)．

通常，最初の数字は伝送路の速度をMbps単位で表しています．10MbpsのEthernetでは10で，1GbpsのGigabit Ethernetは1000Mbpsとして「1000Base……」として表しています．しかし，10Gbpsになると10000となってしまうため，さすがに見にくいものとなり，変えることになりました．しかし10k・Mbpsという解釈もこれまた変なので，「10GBase…」となっています．つまり，従来と同じような表記方法を用いてはいますが，従来のGigabit Ethernetまでが数字＋Mbpsであったことに対して，10Gigabit Ethernetでは数字＋bpsとなっています．

中央の「Base」は相変わらずベースバンド方式のことであり，このあたりに変化はありません(というか変わっていないのはここだけか？)．最後に，「-」の後にインターフェースの特徴を示す記号が入りますが，10Gigabit Ethernetでは三つのパラメータによって示されます．

● 利用する波長の表示

最初が，利用する光の波長を示しており，S，L，Eの3種類があります．このうち，SとLはGigabit Ethernetと同じ短波長(850nm)と，長波長(1310nm)を示します．Eは，これまた長波長に分類されるのですが，1310nmよりさらに波長が長くなる1550nm帯域を利用するものです(**図5**)．

● フレームフォーマットの表示

二つ目のパラメータはフレームフォーマットとして，現実に伝送できるビット量などを示すものとして利用され，X，R，Wがあります．Xは，Gigabit EthernetやFast Ethernetでも利用されていますが，伝送しようとする情報を8ビット-10ビット(8B10B)変換する場合に用いられます．8B10B変換では，8ビットの情報を10ビットのコードに変換して伝送しています．このため，実際に伝送しようとする情報よりも，より多くのビット列を伝送しなければならなくなってしまいます．

実際，Gigabit Ethernetでも最後にXのついている1000Base-LX/SX/CXでは現実の速度は1250Mbpsで伝送されています．ちなみにXのついていない1000Base-Tでは1000Mbpsで伝送されています．

次にRですが，これはXと同じような手法を用いるものの，もう少し効率を良くしたものとなっています．つまりコードへの変換を64B66B変換としているのです．このためXで25％ものオーバヘッドがあったものが，3％強まで減っているので，伝送路の速度をあまり大きくせずに済むようになります．この場合の実際の速度は，10,312.5Mbpsとなっており，64B66B変換したとしても情報の伝送能力は10Gbpsを確保しています．

最後にWですが，10Gigabit Ethernetは拠点間通信を想定している関係上，従来までの伝送設備との整合性も考慮に入れており，Wはそのような設備を使用するために用意されたものです．つまり，SONET系のインターフェースとなっています．SONET系で10Gbpsにもっとも近い9.6Mbpsである，OC-192cのフレームフォーマットを利用します．しかし，OC-192cの9.6Mbpsにはフレームのヘッダ情報を含むので，実際に伝送できる能力は，10Gbpsに届きません．

● 波長多重の有無

最後のパラメータは，そのインターフェースがDWDMを行う際の波長の多重数を示していて，数字が入ります．種類はDWDMを行わず，一つの波長を使っているオーソドックスな場合(シリアルと表現している)．

この場合，数字としては「1」が入ることになるのですが，表記において「1」は省略できるので，DWDMでない場合はインターフェースの種別を示すパラメータは二つだけとなります．実際に数字が入るケースは「4」だけで，それ以外は想定されていません．

〔**図4**〕**10Gigabit Ethernetにおけるインターフェース種別の表現**

〔**図5**〕**利用する波長**

Column
10Gigabit Ethernetのニックネーム？

Ethernetの発展は，10MbpsのEthernetから，10倍である100MbpsのFast Ethernetとなり，さらに10倍の1Gbps（1000Mbps）のGigabit Ethernetが登場しました．この間の頭文字は「E」，「F」，「G」と来たため，続く10Gigabit Ethernetでは，絶対に「H」を頭文字とするニックネームに決まると心から信じていました．しかし結果は，何のひねりもなく「10Gigabit Ethernet」や「10Gig：テンギグ」と呼ばれています．残念です……？

ここの注意点は，四波長多重を行っているからといって，伝送能力が40Gbpsになるわけではありません．じつは逆で，2.5Gbpsを四波長（四色という言い方もする）並列にすることによって，結果10Gbpsの速度を実現しているのです．

● インターフェースの注意点

インターフェース種別を示すパラメータは3種類あり，それぞれが3，3，2個の選択肢をもっているので，そのまま考えれば18種類の組み合わせが考えられます．しかし，実際はそれほど多くはなく，8種類しか規定されていません．一例を示すと，フレームフォーマットに関してですが，64B66Bがあるにも関わらず，8B10Bを用いるメリットとは何でしょうか？

3％そこそこで済むオーバヘッドを25％にすると，その分回線の速度を上げねばならず，実際には12.5Gbpsなどという速度にしなければなりません．したがって，「X」によるフォーマットはなぜあるかといえば，波長多重を行うインターフェースで用いられているのです．

これらの存在は，10Gigabit Ethernetを作り上げる過程で，過去の技術を流用して段階的に実現してきたためにあるのです．このため，現在市場に投入されているインターフェースとしては「S，L，E」と「R，W」の組み合わせと考えて問題ありません（勧告上は，これらにLX4とLW4が加わって8種類となる）．

もう一つの問題は，インターフェースの種類による実質上の速度差です．10Gigabit Ethernetは「10Gbpsのインターフェース」を提供するものです．

10Gigabit Ethernetにおける速度差とは利用するフレームそのものが異なり，それに起因して伝送速度に僅かな差が生じています．よって，10/100Base-TXのようなオートネゴシエーションとは条件が異なります．結論をいえば，10GBase-ERと10GBase-EWでは，同じ1550nm帯域の光を用いてシングルモード光ファイバにより接続される10Gigabit Ethernetのインターフェースであるのですが，通信できません．というかそもそもリンクが成立しません．

当然，10GBase-LRと10GBase-ERという組み合わせにしてフレーム形状を合わせても通信に用いる光の波長が異なるので，こちらもリンクが成立しません．10Gigabit Ethernetのインターフェースは，1文字でも違うものであれば，通信はできないと思ってよいでしょう．

● 関連技術

10GBase-SR/SWは短波長の光を用いるインターフェースですが，使用するケーブルはおもにマルチモード光ファイバケーブル（以降MMF）となっています．高速伝送におけるMMFの問題点に，信号の速度や距離（ケーブル長）の増加によりモード間の到着時間差が大きくなり通信ができなくなるというものがあります．

これはケーブルによって異なり，その特性を「モーダルバンド幅」というパラメータで示しています．この値が大きいMMFほど，高速信号の長距離が可能であるということなのですが，一般に利用されるMMFのモーダルバンド幅は，よいもので500程度であり，10Gbpsの速度では100mの伝送さえできません．

光ファイバケーブルで100mの伝送さえできないというのであれば，お世辞にもあまり魅力のあるインターフェースであるとはいえないものです．しかし，長波長を用いるインターフェースは短波長に比べ高価であるため，安価（あくまで長波長のインターフェースと比較して）な短波長のインターフェースを用いて，それなりの配線を確保するために，現在ではモーダルバンド幅2000というMMFも市場に登場しています．

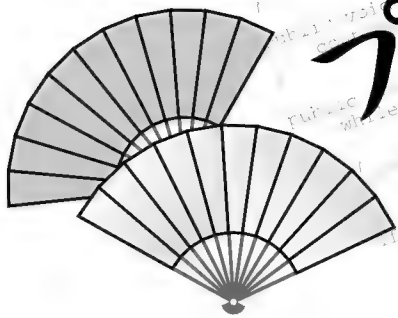
昨年（2002年）5月，アメリカ合衆国Las Vegasにおいて開催されたNetworld+Interop 2002 Las Vegasの隠れた目玉として，マニア受けしていました．ちなみにその直後の6月にAtlantaで開催されたSuperComm 2002 Atlantaでは，これはあまり展示がなく，逆にシングルモード光ファイバケーブルにおいて，1400nm帯に発生する減衰をなくすように加工している，DWDMを意識したものが出展されていました．

まとめ

今回ふれていない1000Base-CXと1000Base-Tのその後ですが，1000Base-CXは10GBase-CX4として，IEEE802.3akで，ツイストペアの1000Base-Tもワークグループは未定ですが，10GBase-Tとして（正気か？!）検討されています．

参考文献

1）松本信幸，『ギガビット・イーサネット絵とき基本用語』，（株）オーム社

まつもと・のぶゆき

宮坂電人

第4回 ハリウッドの法則

フレームワーク

最近よく耳にする言葉の一つに，**フレームワーク(framework)**があります．日本語に翻訳すれば「枠組み」ですが，使われている文脈を読むかぎりでは，どうも枠組みとは違うようです．たとえば，

- これからのWindowsプログラムは.NET Frameworkを使ったものが主流になるらしい
- Mac OS XプログラムではToolboxではなくFrameworkがAPIになるらしい

と語られた場合，「Framework」は「枠組み」を意味しません．その正体は，本連載でよく取り上げるGoF本[注1]で解説されています．すなわち，

- ある特定のソフトウェアを対象にした再利用可能な設計プロダクトを構成するクラスの集合（GoF本の「概論」p.38を参照のこと）

ということです．ようするに，従来は“ライブラリ”と称していたもののオブジェクト指向バージョンではないのかと思った読者もいるでしょう．当たっている部分はありますが，そうでない部分もあります．

逆転した立場

というのも，ライブラリを使うのとフレームワークを使うのでは決定的に異なる側面があります．あるサービスを実現しようと考えた場合，

- ライブラリ——サービスを実現する手続き/関数がないかを探し，見つかればそれらを“呼び出す”よう実装する
- フレームワーク——サービスを実現する手順を探し，見つかればフレームワークから“呼び出される”よう実装する

と，まるで立場が変わったような実装を要求されることがあるからです．

たとえば，ドキュメント（文書）を保存する例で考えましょう．ライブラリを使う場合，ドキュメントを保存するときは，

1) ドキュメントの名前でファイルを作成する手続きを呼び出す
2) ファイルにドキュメントの内容を書き込む
3) ファイルを閉じる手続きを呼び出す

というのが典型パターンでしょう．UNIX，あるいはUNIX系のライブラリを使うなら，

1) `fopen`
2) `fwrite`
3) `fclose`

あたりを利用するでしょう．ところが，文書つきアプリケーションフレームワークを利用する場合は，そう単純ではありません．ドキュメントの内容管理と同時にドキュメントの表示もからみます．ドキュメントを表示するウィンドウの管理もからむので，「独断」でドキュメントの内容を保存できません．保存メニュー，復元メニューなどの許可/不許可の実装[注2]など，とにかく文書つきアプリケーションとしての挙動を首尾一貫させるための“約束ごと”や“世界観”にしばられるわけです．

反面，いったん約束ごとや世界観がわかれば，それらにしたがうのは案外楽なうえに，首尾一貫していることから，さほど苦痛を感じなくなります．しかし，そこに至るまでが苦痛をともないます．プログラマによっては「耐え難い激痛」となり，フレームワークのプログラムについていけず脱落する人もいるようです．

文書つきアプリケーションの約束ごとでは，ドキュメントに関してはドキュメントクラスがあり，独自のドキュメント処理は，ドキュメントクラスを継承した独自のクラスを作成するものが多いようです．ドキュメントの保存は，保存メソッドをオーバライドすることで実装します．つまり「サービスを実現する手順を探し，見つかればフレームワークから“呼び出される”よう実装」とは，

- **サービスを実現する手順**：ドキュメントクラスを継承し，保存メソッドをオーバライドする

注1：『オブジェクト指向における再利用のためのデザインパターン』改訂版，ソフトバンク，ISBN4-7973-1112-6.
注2：たとえばドキュメントの保存前に保存メニューを許可していたなら，保存直後は保存メニューを有効にする意味がないので不許可にする処置など.

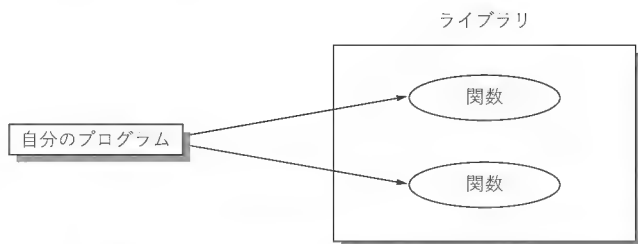
- **フレームワークから呼び出される**：保存メソッドがフレームワークから呼び出される

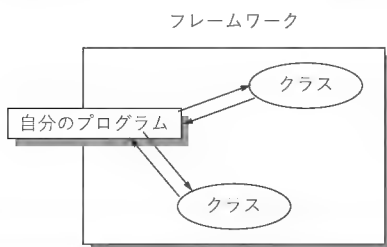
を意味するわけです．ライブラリを呼び出すのとは違い，自分が呼び出される立場に逆転したわけです(**図1**)．クラスライブラリやフレームワークを使ったプログラムが苦手な人たちを観察すると，このような"立場の逆転"にうまく適応できていないように思えてなりません．

ハリウッドの法則

このような，自分が主ではなく従になった状況を表現する用語として"ハリウッドの法則"[注3]があります．これは，映画の都として有名なハリウッドでは脚本や役者が採用されるかどうかはプロデューサに権限があり，脚本家や役者にはないことから来ています．おまけに，採用されたかどうかを脚本家や役者からプロデューサに問い合わせることが許されず，プロデューサから連絡が来るのをひたすら待たねばなりません．

プログラムにおいて，この状況はいうまでもなく，脚本家や役者はプログラマに相当し，プロデューサはクラスライブラリやフレームワークに相当します．プログラムをどう実装するかの主体がプログラマにはなくクラスライブラリやフレームワークにあり，プログラマはひたすらクラスライブラリやフレームワークの約束ごとや世界観にしたがわねばならないのです．これは，従来のプログラムスタイルに慣れたプログラマをとまどわせ，プログラミングの効率が著しく落ち，それまでの自分のキャリアを否定されたかのように感じ，気分を害するプログラマもいます．

〔図1〕立場の逆転

自分のプログラムはライブラリにある関数を呼ぶだけであり，自分のプログラムが主，ライブラリが従の関係になる

(**a**) ライブラリの場合

自分のプログラムはフレームワークを呼ぶだけでなく，逆に呼び出されることがある．またフレームワーク特有の約束ごとや世界観にしばられてしまい，フレームワークが主，自分のプログラムが従の関係になる

(**b**) フレームワークの場合

コールバックと継承，委譲

さきほどの「XXXクラスを継承しYYYメソッドをオーバライドする」という実装方法ですが，フレームワークやクラスライブラリ全般でわりあいよく見かけるパターンです．もちろんフレームワークによっては，継承ではなく委譲を使って実装する例もあります．

いずれにせよ従来の手続き指向ライブラリだと，おそらくコールバック関数を利用して実装することになるでしょう．そう考えると，継承とか委譲とか何やら難しげな用語で説明しなくてもいいのにと思いたいところですが，決定的に違うのは，コールバックは一つの手続き/関数の補足を利用者にやってもらうというオマケ的な意味合いが強いのに対し，継承や委譲では，処理を補足してもらうことが中心になります．補足してもらわねば何もできないという感じです．

コールバックによる実装

一つの例として，ファイルの内容を圧縮してどこかに転送し，転送先で展開するクラスがあったとします．ファイルの読み書きや転送処理はクラスの内部で行うとして，圧縮/展開に関してはクラスの外部で実装してもらいたいとしましょう．圧縮や展開をコールバックでやってもらうという実装例をC++で示すと，**リスト1**のようになります[注4]．

`TransferClass`を利用する例は**リスト2**のようになります．

委譲による実装

委譲とは，自分が実装していない処理を誰かにやってもらうしくみのことをいいます．プログラミング言語の仕様として委譲が規定されている場合もありますが，ライブラリやフレームワークで実装する例が多いでしょう．さきほどの`TransferClass`に委譲処理クラスを加えて実装しなおし，改名すると**リスト3**のようになります．

ここで委譲の対象となるのは，圧縮/展開処理オブジェクト(`CompExpBase`)です．`CompExpBase`は圧縮メソッド/展開メ

注3：Hollywood's law．"ハリウッドの原則"と訳している文献もある．

注4：ふだんならJavaで説明するところだが，Javaではコールバック関数が表現できないのでC++で実装した．なお検証にあたって，筆者が最近利用するようになったMac OS X Ver.10.2.4でProject Builderを利用した．これはgcc version 3.1ベースでコンパイルを行う．

〔リスト1〕コールバックによる実装例

```
class TransferClass {
    std::string mFileName;    //アクセス対象のファイル名
    std::string mChannelName; //転送対象のアドレス名
public:
    //圧縮のコールバック関数の型
    //bool compressF(const void *iData,void *oData,int& ioSize);
    //となる関数を想定する.この関数の引数は
    //iData=圧縮前のデータが格納されている場所
    //oData=圧縮したデータを格納すべき場所
    //ioSize=圧縮前のデータのサイズ,圧縮後はここに圧縮ずみサイズを書き込むこと
    //戻り値はtrueなら成功,falseなら失敗
    typedef bool (*CompressFunc)(const void *,void *,int&);

    //展開のコールバック関数の型
    //bool expandF(const void *iData,void *oData,int iSize);
    //となる関数を想定する.この関数の引数は
    //iData=展開前のデータが格納されている場所
    //oData=展開したデータを格納すべき場所
    //iSize=展開前のデータのサイズ
    //戻り値はtrueなら成功,falseなら失敗
    typedef bool (*ExpandFunc)(const void *,void *,int);

    TransferClass(){
        //...(略)...
    }

    virtual ~TransferClass(){
        //...(略)...
    }

    //転送の準備をする
    //iFileName=アクセス対象のファイル名,iChannelName=転送対象のアドレス名
    //戻り値はtrueなら成功,falseなら失敗
    bool setUp(std::string& iFileName,std::string& iChannelName){
        mFileName = iFileName;
        mChannelName = iChannelName;
        //...(略)...
        return true;
    }
    //ファイルを圧縮して送り出す
    //iCompressF=圧縮のコールバック関数
    //戻り値はtrueなら成功,falseなら失敗
    bool sendData(CompressFunc iCompressF){
        void *aFromBuff; //圧縮前のデータの格納場所
        void *aToBuff;   //圧縮後のデータの格納場所
        int aFromToSize; //圧縮前/後のデータの格納サイズ
        //...(略)...
        //送り出すデータを圧縮する
        if(!iCompressF(aFromBuff,aToBuff,aFromToSize)){
            return false;
        }
        //...(略)...
        return true;
    }

    //圧縮データを受け取りファイルに展開して書き込む
    //iExpandF=展開のコールバック関数
    //戻り値はtrueなら成功,falseなら失敗
    bool recvData(ExpandFunc iExpandF){
        void *aFromBuff; //展開前のデータの格納場所
        void *aToBuff;   //展開後のデータの格納場所
        int aFromSize;   //展開前のデータの格納サイズ
        //...(略)...
        //受け取ったデータを展開する
        if(!iExpandF(aFromBuff,aToBuff,aFromSize)){
            return false;
        }
        //...(略)...
        return true;
    }

    const std::string& getFileName() const {
        return mFileName;
    }

    const std::string& getChannelName() const {
        return mChannelName;
    }
};
```

〔リスト2〕TransferClassを利用する例

```
static bool compressFunc(const void *iData,void *oData,int& ioSize)
{
    //...(略)...
    return true;
}

static bool expandFunc(const void *iData,void *oData,int iSize)
{
    //...(略)...
    return true;
}

static void Test()
{
    TransferClass aTC;
    std::string aSendFile("./sendtest.bin");
    std::string aRecvFile("./recvtest.bin");
    std::string aChannel("192.168.1.2");

    //送り出す例
    if(aTC.setUp(aSendFile,aChannel)){
        if(aTC.sendData(compressFunc)){
            std::cout << "file:" << aTC.getFileName() << " was send to " << aTC.getChannelName() << ".¥n";
        }else{
            std::cout << "error:send sendData¥n";
        }
    }else{
        std::cout << "error:send setUp¥n";
    }

    //受け取る例
    if(aTC.setUp(aRecvFile,aChannel)){
        if(aTC.recvData(expandFunc)){
            std::cout << "file:" << aTC.getFileName() << " was received from " << aTC.getChannelName() << ".¥n";
        }else{
            std::cout << "error:receive recvData¥n";
        }
    }else{
        std::cout << "error:receive setUp¥n";
    }
}
```

〔リスト3〕委譲による実装例

```
class CompExpBase { //委譲オブジェクトのベースクラス(抽象クラス)
public:
    //圧縮のメソッド
    //iData=圧縮前のデータが格納されている場所
    //oData=圧縮したデータを格納すべき場所
    //ioSize=圧縮前のデータのサイズ,圧縮後はここに圧縮ずみサイズを書き込むこと
    //戻り値はtrueなら成功,falseなら失敗
    virtual bool compressMethod(const void *iData,void *oData,
                                                  int& ioSize) = 0;

    //展開のメソッド
    //iData=展開前のデータが格納されている場所
    //oData=展開したデータを格納すべき場所
    //iSize=展開前のデータのサイズ
    //戻り値はtrueなら成功,falseなら失敗
    virtual bool expandMethod(const void *iData,void *oData,
                                                  int iSize) = 0;
};

class TransDeleClass {
    std::string mFileName;    //アクセス対象のファイル名
    std::string mChannelName; //転送対象のアドレス名
    CompExpBase *mDelegate;   //処理を委譲するオブジェクト

    TransDeleClass(); //(空っぽ,パラメータなしのコンストラクタ起動防止用)
public:
    TransDeleClass(CompExpBase *iDelegate){
        mDelegate = iDelegate;
        //...(略)...
    }

    virtual ~TransDeleClass(){
        //...(略)...
    }

    //転送の準備をする
    //iFileName=アクセス対象のファイル名,iChannelName=転送対象のアドレス名
    //戻り値はtrueなら成功,falseなら失敗
    bool setUp(std::string& iFileName,std::string& iChannelName){
        mFileName = iFileName;
        mChannelName = iChannelName;
        //...(略)...
        return true;
    }

    //ファイルを圧縮して送り出す
    //iCompressF=圧縮のコールバック関数
    //戻り値はtrueなら成功,falseなら失敗
    bool sendData(){
        void *aFromBuff; //圧縮前のデータの格納場所
        void *aToBuff;   //圧縮後のデータの格納場所
        int aFromToSize; //圧縮前/後のデータの格納サイズ
        //...(略)...
        //送り出すデータを圧縮する
        if(!mDelegate->compressMethod(aFromBuff,aToBuff,
                                              aFromToSize)){
            return false;
        }
        //...(略)...
        return true;
    }

    //圧縮データを受け取りファイルに展開して書き込む
    //iExpandF=展開のコールバック関数
    //戻り値はtrueなら成功,falseなら失敗
    bool recvData(){
        void *aFromBuff; //展開前のデータの格納場所
        void *aToBuff;   //展開後のデータの格納場所
        int aFromSize;   //展開前のデータの格納サイズ
        //...(略)...
        //受け取ったデータを展開する
        if(!mDelegate->expandMethod(aFromBuff,aToBuff,
                                              aFromSize)){
            return false;
        }
        //...(略)...
        return true;
    }
    ...(略)...
};
```

ソッドのインターフェースのみを定義する抽象クラスです．したがって実際に圧縮/展開を行わせるためには，CompExpBaseを継承して具象クラスを作り，未実装になっているメソッド(compressMethod，expandMethod)を記述する必要があります．さらにそのクラスで発生したインスタンスを引き数にして，TransDeleClassインスタンスを作成します．以上の実装を行った例は，**リスト4**のようになります．

継承による実装

一方，フレームワークやクラスライブラリで見かける実装だと，

- **提供側**：圧縮/展開メソッドを未実装にしたファイル圧縮/展開転送クラス(抽象クラス)を用意する
- **利用側**：ファイル圧縮/展開転送クラスを使いたいなら，このクラスを継承し未実装メソッドを実装する

という役割分担になります．この考えで最初のTransferClassを実装しなおし，改名すると**リスト5**のようになります．

見てわかるとおり，圧縮/展開メソッド(compressF，expandF)が純粋仮想関数すなわち未実装になっています．このままではTransferBaseは使えないので，これを継承したクラスを作成して対応した例は**リスト6**のようになります．

見方によっては，コールバック関数だった箇所を純粋仮想関数に書き換えただけ，あるいは委譲クラスとして分離していたものを一つのクラスに一体化しただけです．しかし未実装箇所が一つのクラスにまとまったこと，クラスやインスタンスの数が減ることで多少見通しがよくなる効果が期待できます．

Template Methodパターン

ここで例題として出したCompExpBase，TransferBaseのような「未実装メソッドがある抽象クラスを提供し，利用する側はそのクラスを継承して未実装メソッドをオーバライドして利用する」パターンを，GoF本ではTemplate Methodパターンとして紹介しています(**図2**)．

GoF本では，Template Methodパターンを次のように説明しています[注5]．

> 一つのオペレーションにアルゴリズムのスケルトンを定義しておき，その中のいくつかのステップについては，サブクラスでの定義にまかせる．

注5：GoF本のTemplate Methodの「目的」より引用．p.347．

〔リスト4〕TransDeleClassを利用する例

```
class MyCompExp : public CompExpBase { //委譲オブジェクトのクラス(具象クラス)
public:
    bool compressMethod(const void *iData,void *oData,int& ioSize)
    {
        //...(略)...
        return true;
    }

    bool expandMethod(const void *iData,void *oData,int iSize)
    {
        //...(略)...
        return true;
    }
};

static void Test()
{
    MyCompExp aDelegate; //委譲オブジェクト
    TransDeleClass aTC(&aDelegate);
    std::string aSendFile("./sendtest.bin");
    std::string aRecvFile("./recvtest.bin");
    std::string aChannel("192.168.1.2");

    //送り出す例
    if(aTC.setUp(aSendFile,aChannel)){
        if(aTC.sendData()){
            std::cout << "file:" << aTC.getFileName() << " was send to " << aTC.getChannelName() << ".¥n";
        }else{
            std::cout << "error:send sendData¥n";
        }
    }else{
        std::cout << "error:send setUp¥n";
    }

    //受け取る例
    if(aTC.setUp(aRecvFile,aChannel)){
        if(aTC.recvData()){
            std::cout << "file:" << aTC.getFileName() << " was received from " << aTC.getChannelName() << ".¥n";
        }else{
            std::cout << "error:receive recvData¥n";
        }
    }else{
        std::cout << "error:receive setUp¥n";
    }
}
```

〔リスト5〕継承による実装例

```
class TransferBase { //ファイル圧縮/展開転送のベースクラス(抽象クラス)
    std::string mFileName;    //アクセス対象のファイル名
    std::string mChannelName; //転送対象のアドレス名
protected:
    //圧縮のメソッド
    //iData=圧縮前のデータが格納されている場所
    //oData=圧縮したデータを格納すべき場所
    //ioSize=圧縮前のデータのサイズ,圧縮後はここに圧縮ずみサイズを書き込むこと
    //戻り値はtrueなら成功,falseなら失敗
    virtual bool compressF(const void *iData,void *oData,int& ioSize) = 0;

    //展開のメソッド
    //iData=展開前のデータが格納されている場所
    //oData=展開したデータを格納すべき場所
    //iSize=展開前のデータのサイズ
    //戻り値はtrueなら成功,falseなら失敗
    virtual bool expandF(const void *iData,void *oData,int iSize) = 0;

public:
    TransferBase(){
        //...(略)...
    }

    virtual ~TransferBase(){
        //...(略)...
    }

    //転送の準備をする
    //iFileName=アクセス対象のファイル名,iChannelName=転送対象のアドレス名
    //戻り値はtrueなら成功,falseなら失敗
    bool setUp(std::string& iFileName,std::string& iChannelName){
        mFileName = iFileName;
        mChannelName = iChannelName;
        //...(略)...
        return true;
    }

    //ファイルを圧縮して送り出す
    //戻り値はtrueなら成功,falseなら失敗
    bool sendData(){
        void *aFromBuff; //圧縮前のデータの格納場所
        void *aToBuff;   //圧縮後のデータの格納場所
        int aFromToSize; //圧縮前/後のデータの格納サイズ
        //...(略)...
        //送り出すデータを圧縮する
        if(!compressF(aFromBuff,aToBuff,aFromToSize)){
            return false;
        }
        //...(略)...
        return true;
    }

    //圧縮データを受け取りファイルに展開して書き込む
    //戻り値はtrueなら成功,falseなら失敗
    bool recvData(){
        void *aFromBuff; //展開前のデータの格納場所
        void *aToBuff;   //展開後のデータの格納場所
        int aFromSize;   //展開前のデータの格納サイズ
        //...(略)...
        //受け取ったデータを展開する
        if(!expandF(aFromBuff,aToBuff,aFromSize)){
            return false;
        }
        //...(略)...
        return true;
    }
    ...(略)...
};
```

〔リスト6〕TransferBaseを利用する例

```
class MyTransfer : public TransferBase {
//ファイル圧縮/展開転送のクラス(具象クラス)
protected:
    bool compressF(const void *iData,void *oData,int& ioSize)
    {
        //...(略)...
        return true;
    }

    bool expandF(const void *iData,void *oData,int iSize)
    {
        //...(略)...
        return true;
    }
};

static void Test()
{
    MyTransfer aTC;
    std::string aSendFile("./sendtest.bin");
    std::string aRecvFile("./recvtest.bin");
    std::string aChannel("192.168.1.2");

    //送り出す例
    if(aTC.setUp(aSendFile,aChannel)){
        if(aTC.sendData()){
            std::cout << "file:" << aTC.getFileName()
             << " was send to " << aTC.getChannelName() << ".¥n";
        }else{
            std::cout << "error:send sendData¥n";
        }
    }else{
        std::cout << "error:send setUp¥n";
    }

    //受け取る例
    if(aTC.setUp(aRecvFile,aChannel)){
        if(aTC.recvData()){
            std::cout << "file:" << aTC.getFileName()
       << " was received from " << aTC.getChannelName() << ".¥n";
        }else{
            std::cout << "error:receive recvData¥n";
        }
    }else{
        std::cout << "error:receive setUp¥n";
    }
}
```

〔図2〕Template Methodパターン

残念ながらこの説明は難解で，誤解を生みやすくなっています．恥ずかしながら筆者も，この説明がさっぱり理解できなかった一人です．じつは要点は“サブクラスでの定義にまかせる”にあります．表現を変えると，“提供するベースクラスでは詳細に定義しない”ともいえます．GoF本のTemplate Methodパターンの「適用可能性」を読めば，もう少しはっきりしてきます[注6]．

> アルゴリズムの不変な部分をまず実装し，ふるまいが変わり得る部分の実装はサブクラスに残しておく．

例題のTransferBaseで説明しましょう．「アルゴリズムの不変な部分をまず実装」というのはsetUp，sendData，recvDataを示しています．「ふるまいが変わり得る部分」とは，純粋仮想関数であるcompressF，expandFです．

ふるまいが変わり得る圧縮/展開をベースクラスで実装しなかったことにより，いろいろな圧縮/展開アルゴリズムが採用できるというメリットがあります．圧縮/展開はいろいろなアルゴリズムがあり，圧縮率は良いが処理速度が遅いものもあれば，圧縮率は悪いが処理速度が速いものもあり，状況に応じて使い分けたい場合があります．また，せっかく良いアルゴリズムであっても特許が取得されているために使えなくなるケースもあります．こういった状況があるため，アルゴリズムが簡単に変更できるとありがたいわけです[注7]．

Template Methodの応用パターン

GoF本では23のデザインパターンが紹介されていますが，面白いことにほかのデザインパターンを観察すると，Template Methodに通じるクラスが出現するパターンがあります．つまり「未実装メソッドがある抽象クラス」と「そのメソッドを実装した子クラス(具象クラス)」が登場するパターンです．次がそのパターンです．

> Abstract Factory，Builder，Factory Method，Prototype，Bridge，Composite，Decorator，Flyweight，Proxy，Chain of Responsibility，Command，Interpreter，Iterator，Mediator，Observer，State，Strategy，Visitor

驚いたことに，かなりのパターンに見当たります．ひょっとすると「未実装メソッドがある抽象クラス」，「メソッドを実装した子クラス」の組み合わせは，オブジェクト指向プログラムでは基本的な戦術なのかもしれません．

じつは，この推測を裏づけるような“原則(Principle)”があるのですが，あまり知られていません．次回は，この原則について取り上げたいと思います．

みやさか・でんと　miyadent@anet.ne.jp

注6：GoF本のTemplate Methodの「適用可能性」より引用．p.348．
注7：アルゴリズムの変更が主体である場合は，Template MethodパターンよりもStrategyパターンが見出しやすくなる．

XScaleプロセッサ徹底活用研究

第2回　XScaleプロセッサのプログラミング

山本繁寿

前回は，CQ RISC評価キット/XScaleに搭載されているPXA250のハードウェアと，本評価キットのメモリマップなどについて解説した．2回目の今回は，この上で動作するXScale用サンプルプログラムについて解説する．まずはCPUボード上の7セグメントLEDを点灯制御するもっとも簡単なプログラムを，そしてタイマ割り込みまたはプッシュスイッチによる外部割り込み制御プログラムを解説する．最後に，UART制御プログラムや，SDRAMのテストプログラムなども解説する．（編集部）

はじめに

CQ RISC評価キット/XScaleに搭載されているPXA250には，さまざまな周辺機能が内蔵されています．そのすべてを解説することはできませんが，ここでは次のようなもっとも基本的なサンプルについて，いくつか解説していきます．

(1) LED点灯制御プログラム
(2) タイマ割り込みプログラム
(3) プッシュスイッチによる外部割り込みのプログラム
(4) シリアル通信プログラム
(5) RAMテストプログラム
(6) CPUモード&キャッシュ設定プログラム

1　LED点灯制御プログラム

評価ボードには，赤色および緑色のLEDと，7セグメントLEDがそれぞれ実装されています．ここでは，各LEDを点灯させるプログラムについて解説します．

● CqREEK/XScale CPUボードの仕様

前回のハードウェア編で解説したように，本CPUボード搭載のLED1（赤）とLED2（緑）および7セグメントLEDは，CS2の空間にマッピングしたポート機能で制御します．アドレス定義のヘッダファイルを**リスト1**（**a**）に示します．

● LED1およびLED2の点灯制御

これらのLEDの点灯制御レジスタは，コンフィグレーションポート（アドレス0x0A000000）にマッピングされています．LED1を点灯させる場合は，コンフィグレーションポートのビット12を0，逆に消灯させる場合はビット12に1を書き込みます．LED2を点灯制御する場合は，ビット13に対して0または1を書き込みます．

リスト1（**b**）のⓐとⓑは，変数led_countのビット0とビット1の状態によって，LED1とLED2を点灯制御している例です．

● 7セグメントLED

7セグメントLEDは，7セグメントLEDポート（アドレス0x0A000010）にマッピングされています．7セグメントLEDの各セグメント（ビット）に0を書き込むと点灯，1を書き込むと消灯します．

サンプルプログラムでは，定数を配列で定義し，‘0’〜‘F’までの16進数の点灯パターンを定義しています〔**リスト1**（**b**）の

〔**リスト1**〕**LED点灯制御プログラム**

```
#ifndef _SAMPLE_H_
#define _SAMPLE_H_
#define _ulong unsigned long
〜 中略 〜

/* CqREEK/XScale CPUボード ポート制御レジスタ */
#define  CONFIG    (volatile ulong *)0x0a000000
#define  LEDPORT   (volatile ulong *)0x0a000010
#define  DIPSWIN   (volatile ulong *)0x0a000018

#endif
```

（**a**）sample.h

```
#include "sample.h"

/* 7セグメントLED点灯パターン('0' to 'F' & ' ') */
char     led_7seg[]={0xc0,0xF9,0xa4,0xb0,0x99,0x92,0x82,
         0xd8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e,0xFF};

int main(void)
{
    int i,led_count;

    /* LEDカウントクリア&LED表示クリア */
    led_count=0;
    *LEDPORT = led_7seg[led_count];
    *CONFIG  |= 0x3000;            /* LED1/2消灯 */

    while(1){
         for(i=0;i<0x40000;i++){}/* ウェイト */
         led_count++;             /* LEDカウント+1 */
         if (led_count>15) led_count=0;
         if (led_count&1) {       /* ビット0='1' */
              *CONFIG &= 0xefff;  /* LED1点灯 */
         } else {                 /* ビット0='0' */
              *CONFIG |= 0x1000;  /* LED1消灯 */
         }
         if (led_count&2) {       /* ビット1='1' */
              *CONFIG &= 0xdfff;  /* LED2点灯 */
         } else {                 /* ビット1='0' */
              *CONFIG |= 0x2000;  /* LED2消灯 */
         }
         /* LED点灯制御 */
         *LEDPORT = led_7seg[led_count];
    }
}

int atexit()
{
    for(;;){}
}
```

ⓐLED1の点灯制御
ⓑLED2の点灯制御
ⓒ7セグメントLED点灯パターン
ⓓ7セグメントLED2の点灯制御

（**b**）led.c

ⓒ〕．そして変数led_countの値によって，7セグメントLEDに表示する点灯パターンを，7セグメントLEDポートに書き込みます〔**リスト1(b)**のⓓ〕．

なお，**リスト1(b)**の最後にatexit()という関数がありますが，これはCプログラムのmain()関数の実行が終了した後に実行する処理を記述します．しかし本評価キットの環境にはmain()関数が戻るべき場所，つまりOSなどは用意していないので，main()関数は必ず無限ループするように記述してください．またatexit()を記述しないとリンク時にエラーが出るので，ダミーの関数を記述しておいてください．

ちなみに，リンク時のリンクアドレスなどの指定やスタートアップルーチンは，評価キット添付のサンプルプログラムのものを流用しています．

● ディップスイッチの状態入力

評価ボード上には8ビットのディップスイッチがあり，そのうち6ビットをユーザーが自由に使うことができます．

リスト1のプログラムでは使用していませんが，ディップスイッチの状態を取得する場合は，次のようにします．

```
c = *DIPSWIN;
```

2 タイマ割り込みプログラム

ここでは，リアルタイムクロック(以下RTC)を利用して，一定時間が経過したらタイマ割り込みを発生させ，7セグメントLEDをカウントアップ表示するプログラムについて解説します．プログラム全体の流れは，次のようになります．

(1) RTCレジスタの初期化
(2) 割り込みコントローラの初期化
(3) 割り込みが発生したら7セグメントLEDの点灯

● リアルタイムクロックレジスタの初期化

RTCレジスタの初期化は，次の手順で行います〔**リスト2(a)**のⓐ〕．

▶RTC割り込みを禁止(RTSR)

RTCレジスタ初期化中に割り込みを発生させないようにするために，RTC Statusレジスタ(RTSR，アドレス0x40900008)のすべてのビットを0にして，割り込みを禁止します．

▶コンペアカウンタの設定(RTAR)

RTC Alarmレジスタ(RTAR，アドレス0x40900004)に設定した値は，コンペアカウンタとして使用されます．RTARレジスタに設定した値は，次に設定するRCNRレジスタの値と比較されます．RCNRレジスタの値とRTARレジスタの値が同じになり，かつRTSRレジスタのALEビット(RTC割り込みイネーブルビット)がセットされていると，RTSRレジスタのAL(RTC割り込み検出ビット)がセットされます．これにより，RTCを利用したタイマ割り込みが発生するようになります．

▶RTCカウンタ値の設定(RCNR)

RTARレジスタで設定した値と比較するために，RTC Counterレジスタ(RCNR，アドレス0x40900000)を設定します．

▶RTC割り込みを許可(RTSR)

RTCによるタイマ割り込みを発生させるために，RTSRのALE(ビット2)をセットして，割り込みを有効にします．

● 割り込みコントローラの初期化

PXA250の割り込みコントローラの初期化は，次の三つのレジスタを初期化する必要があります〔**リスト2(a)**のⓑ〕．

- 割り込みソースの設定(ICMR)
- 割り込みの種類の設定(ICLR)
- IDLEモード中の割り込みの設定(ICCR)

個々のレジスタの設定について解説します．

▶割り込みの種類の設定(ICLR)

PXA250はARMアーキテクチャのCPUです．ARMアーキテクチャには割り込みとして，IRQ割り込み，FIQ割り込み，ソフトウェア割り込みの3種類があります．ここでは，タイマ割り込み(RTC割り込み)をIRQ割り込みとして設定するために，Interrupt Controller Levelレジスタ(ICLR，アドレス0x40D00008)のIL31(ビット31)を0にクリアします．

▶IDLEモード中の割り込みの設定(ICCR)

PXA250はIDLEモードになるとCPUコアクロックは停止します．IDLEモード中にすべての有効な割り込みをCPUに入れるためには，Interrupt Controller Controlレジスタ(ICCR，アドレス0x40D00014)のDIM(ビット0)を0にクリアします．

▶割り込みソースの設定(ICMR)

PXA250の割り込みには，シリアルやUSBなどのCPU内蔵デバイスによる割り込み，外部に接続したデバイスからGPIOを使って割り込みを入力する外部割り込みなど，合計22種類の割り込み要因があります．ここではタイマ割り込みを使うので，割り込み要因としてリアルタイムクロック(RTC)を使用するために，Interrupt Controller Maskレジスタ(ICMR，アドレス0x40D00004)のIM31(ビット31)を1にセットします．

● 割り込みハンドラ

ARMアーキテクチャの本来の割り込みベクタは，0x0番地から始まります．しかし評価ボードではこのアドレスにはフラッシュメモリを実装しています．ROMは書き換えることができないので，このままでは，ユーザーが自由に割り込み処理プログラムを書くことができません．

そこで評価ボードでは，RAM上に代替割り込みベクタを用意し，本来の割り込みベクタから代替割り込みベクタにジャンプするようにしています．よってユーザーが割り込み処理を記述する場合は，代替割り込みベクタを使います．評価ボードに付属したモニタデバッガを使用する場合，代替割り込みベクタは0xA0000000からのアドレスに配置しています．

リスト2(b)に代替割り込みベクタや割り込みハンドラのソースを示します．割り込みハンドラでは，スタックやコンテキストなどの保存，CPUモードの設定，多重割り込みへの対応など，割り込み処理の前処理を行います．

〔リスト2〕RTCによるタイマ割り込みプログラム

```
volatile int timer_count;
～ 中略 ～
int main(void)
{
    int i,led_count;

    /* RTC初期化 */
    *RTSR = 0x0001;     /* intrrupt disable */
    *RTAR = 0x0002;     /* compare count set */
    *RCNR = 0x0000;     /* count 0 clear */
    *RTSR = 0x0004;     /* intrrupt enable */

    /* 割り込みコントローラ初期化 */
    *ICLR &= 0x7FFFFFFF;     /* RTC=IRQ */
    *ICCR = 0x00000000;
    *ICMR |= 0x80000000;     /* RTC割り込み使用 */

    /* RTC割り込みカウンタクリア */
    i=timer_count=0;

    /* LEDカウントクリア&LED表示クリア */
    led_count=0;
    *LEDPORT = led_7seg[led_count];

    while(1){
         /* RTC割り込みが発生したか確認 */
         if (i!=timer_count) {
              led_count++;
              if (led_count>15) led_count=0;
              /* LED点灯制御 */
              *LEDPORT = led_7seg[led_count];
              /* 現在のカウント値を保存 */
              i=timer_count;
         }
    }
}
～ 以下略 ～
```

ⓐRTCの初期化

ⓑ割り込みコントローラの初期化

ⓒ割り込みが発生したら7セグメントLEDをカウントアップ表示

（a） timer.c

```
#include "sample.h"
～ 中略 ～

extern volatile int    timer_count;
～ 中略 ～

void     SS_Irq(void)
{
    /* RTC割り込み発生 */
    if((*ICIP & 0x80000000) == 0x80000000) {
         timer_count++;
         *RCNR = 0x0000;     /* count 0 clear */
         *RTSR = 0x0005;     /* clear AL bit */
    }

    /* 他に割り込み処理があれば記述 */
}
```

（c） exception.c

```
    .text

    .extern        main
    .extern        SS_SoftwareInterrupt
    .extern        SS_AbortPrefetch
    .extern        SS_AbortData
    .extern        SS_Reserved
    .extern        SS_Irq

    B    _startup
    nop
    B    _SS_SoftwareInterrupt
    B    _SS_AbortPrefetch
    B    _SS_AbortData
    B    _SS_Reserved
    B    _SS_Irq_asm
    NOP
    NOP
    NOP
    NOP

_startup:

#CPSR Fbit,Ibit enable

    MRS  R1,cpsr
    BIC  R1,R1,#0xC0
    MSR  cpsr,R1

#IRQ Mode
    MRS  R1,cpsr
    BIC  R1,R1,#0x01
    MSR  cpsr,R1

#IRQ Mode の Stack 設定
    LDR  R13,=0xA0FE0000

#SVC Mode に戻す
    MRS  R1,cpsr
    ORR  R1,R1,#0x01
    MSR  cpsr,R1

#jump to main
    BL   _main
    NOP

_SS_Irq_asm:

# 前処理
    STMFD     sp!,{r0-r12,lr}
    MRS  r10,spsr
    STMFD     sp!,{r10}

# 割り込み処理に分岐
    bl   _SS_Irq

# 後処理
    LDMFD     sp!,{r11}
    MSR  spsr_c,r11
    LDMFD     sp!,{r0-r12,lr}

    SUBS pc,lr,#4
    nop
    nop
    nop
```


（b） startup.s

本評価キットはモニタ型デバッガなので，デバッガが起動状態にあるときは実際にはすでにCPUが稼動状態にあるわけですが，本評価キットではユーザープログラムの実行開始時，この代替割り込みベクタのリセットベクタから実行を開始するようになっているので，ここにスタートアップルーチンの記述も必要になります．つまり，**リスト2(b)**は割り込み処理対応のstartup.sというわけです．

● 割り込み処理

RTCによるタイマ割り込みが発生すると，本来の割り込みベクタから代替割り込みベクタにジャンプし，**リスト2(b)**に示すように，SS_IrqというCの関数にジャンプするようになっています．

リスト2(c)に実際の割り込み処理の内容を示します．まずICIPレジスタを読み出し，発生した割り込みがRTC割り込みかどうかを確認します．RTC割り込みだった場合は，割り込みカウント変数をインクリメントし，RTCレジスタをアクセスしてカウンタと割り込み要求をクリアして戻ります．

実際に7セグメントLEDを点灯制御する部分は，**リスト2**

〔リスト3〕プッシュスイッチによる外部割り込みプログラム

```
#include "sample.h"
volatile int gpio_count;
volatile int mode;
〜 中略 〜

int main(void)
{
    int i,led_count;

    mode = *DIPSWIN & 0x2;  /* 接続方式を検出 */

    switch(mode){
        case 0x0: /* USB接続 */
            /* GPIO(GP1)割り込み初期化 */
            *GRER_x = *GRER_x & 0xFFFFFFFD;
            /* GP1立ち下がりエッジ検出 */
            *GFER_x = *GFER_x | 0x00000002;
            /* GP1入力方向 */
            *GPDR_x = *GPDR_x & 0xFFFFFFFD;
            /* GP1エッジ検出フラグクリア */
            *GEDR_x = 0x00000002;

            /* 割り込みコントローラ初期化 */
            /* GP1=IRQ */
            *ICLR &= 0xFFFFFDFF;
            *ICCR = 0x00000000;
            /* GP1 割り込み使用 */
            *ICMR |= 0x00000200;
            break;

        case 0x2: /* シリアル接続 */
            /* GPIO(GP0)割り込み初期化 */
            /* GP0立ち上がりエッジ検出 */
            *GRER_x = *GRER_x & 0x00000001;
            *GFER_x = *GFER_x & 0xFFFFFFFE;
            /* GP0入力方向 */
            *GPDR_x = *GPDR_x & 0xFFFFFFFE;
            /* GP0エッジ検出フラグクリア */
            *GEDR_x = 0x00000001;

            /* 割り込みコントローラ初期化 */
            /* GP0=IRQ */
            *ICLR &= 0xFFFFFEFF;
            *ICCR = 0x00000000;
            /* GP0 割り込み使用 */
            *ICMR |= 0x00000100;
            break;

        default:
            break;
    }

    /* GPIO割り込みカウンタクリア */
    i=gpio_count=0;

    /* LEDカウントクリア&LED表示クリア */
    led_count=0;
    *LEDPORT = led_7seg[led_count];

    while(1){
        /* GPIO割り込みが発生したか確認 */
        if (i!=gpio_count) {
            led_count++;
            if (led_count>15) led_count=0;
            /* LED点灯制御 */
            *LEDPORT = led_7seg[led_count];
            /* 現在のカウント値を保存 */
            i=gpio_count;
        }
    }
}
〜 以下略 〜
```

ⓐ初期化部分

(a) gpio.c

```
#include "sample.h"
〜 中略 〜

extern volatile int    timer_count;
〜 中略 〜

void    SS_Irq(void)
{
    switch(mode){
        case 0x0: /* USB接続 */
            /* GPIO(GP1)割り込み発生 */
            if((*ICIP & 0x00000200) == 0x00000200) {
                /* GP1 立ち下がりエッジ検出 */
                if (*GEDR_x & 0x00000002) {
                    /* 検出フラグクリア */
                    *GEDR_x = 0x00000002;
                    gpio_count++;
                }
            }
            break;

        case 0x2: /* シリアル接続 */
            /* GPIO(GP0)割り込み発生 */
            if((*ICIP & 0x00000100) == 0x00000100) {
                /* GP0 立ち上がりエッジ検出 */
                if (*GEDR_x & 0x00000001) {
                    /* 検出フラグクリア */
                    *GEDR_x = 0x00000001;
                    gpio_count++;
                }
            }
            break;

        default:
            break;
    }

    /* 他に割り込み処理があれば記述 */
}
```

(b) exception.c

(a)のⓒに示すようにメインルーチンに置いています．保持してある変数と割り込みカウント変数を比較し，一致しなければ割り込みが発生したことになるので，7セグメントLEDをカウントアップ表示するという内容です．

リスト2のプログラムではRTARレジスタに2を設定しているので，約2秒に1回，7セグメントLEDの表示がカウントアップします．

3 プッシュスイッチによる外部割り込みプログラム

● プッシュスイッチの仕様

次は，プッシュスイッチによる外部割り込みのプログラムについて解説します．評価ボードには，次の3種類のプッシュスイッチが実装されています．

- RESETスイッチ ……リセット入力
- BRKRQスイッチ ……GP1の入力
- GP0スイッチ…………GP0の入力

RESETスイッチを押すとCPUボード全体にリセットがかかるため，一般的な外部割り込みの評価には適していません．そこで残るのは，BRKRQスイッチとGP0スイッチです．

本評価ボードはデバッガとの接続にシリアル接続とUSB接続を選択できますが，CPUボードの仕様上，シリアル接続時はGP1入力を強制ブレーク機能に，USB接続時はGP0をUSB電源検出用に使っています．

よって，JCOM2を使わない場合はGP1につながっているBRKRQスイッチが，USBを使わない場合はGP0につながっているGP0スイッチが，ユーザーが自由に使えるGP入力スイッチとなります．

なお，ボードの仕様上，GP0スイッチを押すと“H”レベルが，BRKRQスイッチは押すと“L”レベルが入力されます．よって検出するエッジが逆になるので注意してください．

プログラミングの手順は，次のようになります．

▶USB接続時

(1) GP1の立ち上がりエッジ検出を無効(GRER0)
(2) GP1の立ち下がりエッジ検出を有効(GFER0)
(3) GP1を入力に設定(GPDR0)
(4) GP1のエッジ検出フラグをクリア(GEDR0)
(5) GP1の割り込みをIRQに設定(ICLR)
(6) IDLEモード中の割り込みの設定(ICCR)
(7) GP1の割り込みを許可(ICMR)

▶シリアル接続時

(1) GP0の立ち上がりエッジ検出を有効(GRER0)
(2) GP0の立ち下がりエッジ検出を無効(GFER0)
(3) GP0を入力に設定(GPDR0)
(4) GP0のエッジ検出フラグをクリア(GEDR0)
(5) GP0の割り込みをIRQに設定(ICLR)
(6) IDLEモード中の割り込みの設定(ICCR)
(7) GP0の割り込みを許可(ICMR)

次に，初期化の手順を説明します〔**リスト3(a)**のⓐ〕．

● GPIOの初期化

▶GP0/GP1の立ち上がりエッジ検出(GRER0)

シリアル接続時は，GP0の立ち上がりエッジを検出するので，GPIO Rising Edge detect Enableレジスタ0(GRER0，アドレス0x40E00030)のビット0をセットしておきます．

USB接続時はGP1の立ち下がりエッジを検出するので，立ち上がりエッジを検出するGRER0のビット1はクリアしておきます．

▶GP0/GP1の立ち下がりエッジ検出(GFER0)

USB接続時はGP1の立ち下がりエッジを検出するので，GPIO Falling Edge detect Enableレジスタ0(GFER0レジスタ，アドレス0x40E0003C)のビット1はセットしておきます．

シリアル接続時はGP0の立ち上がりエッジを検出するので，立ち下がりエッジを検出するGFER0のビット0はクリアしておきます．

▶GP0/GP1を入力に設定(GPDR0)

GPIO Pin Direction レジスタ0(GPDR0レジスタ，アドレス0x40E0000C)のPD0(ビット0)またはPD1(ビット1)を0に設定することで，GP0/GP1を入力に設定します．

▶GP0/GP1のエッジ検出フラグをクリア(GEDR0)

GPIO Pin Direction レジスタ0(GEDR0レジスタ，アドレス0x40E00048)のED0(ビット0)またはED1(ビット1)に1を書き込み，すでにエッジ検出フラグが立っていればクリアしておきます．

▶GP0/GP1の割り込みをIRQに設定(ICLR)

GP0/GP1の割り込みをIRQで使うので，ICLRのビット8またはビット9をクリアします．

▶IDLEモード中の割り込みの設定(ICCR)

これは，先ほどのタイマ割り込みの例と同じです．

▶GP0/GP1の割り込みを許可(ICMR)

GP0/GP1からの割り込みを許可するために，ICMRのビット8またはビット9をセットします．

● 割り込み処理

リスト3(b)に実際の割り込み処理関数を示します．ICIPのビット8またはビット9をチェックして，GP0またはGP1でGPIO割り込みが発生したことを判定します．GP0またはGP1からの割り込みであることが確認できたら，GP0またはGP1のエッジ検出フラグをクリアして割り込み要求をクリアし，割り込みカウント変数をインクリメントして処理を終わります．

実際に7セグメントLEDを点灯制御する部分は，さきほどのタイマ割り込みと同じです．

GP0およびGP1はデバッガも使っているので，各レジスタは使用するリソース以外の設定は状態を保存するように気をつけてください．

4 シリアル通信プログラム

次は，シリアルポートを使った通信プログラムを作成してみます．

● FFUARTとGPIO

PXA250のGPIOは，GP0～GP80までの81本あります．このなかでGP34～GP41は，Alternate FunctionとしてFFUARTの機能と兼用ピンになっています．リセット直後のデフォルト設定では，GPIOとして動作しています．

また評価ボードにはシリアルポートとしてJCOM1とJCOM2の2ポート実装されています．JCOM1はCPUのFFUARTに，JCOM2はCPUのBTUARTに接続されています．JCOM2はデバッガとの接続でシリアル接続を選択した場合に使うポートなので，ここではJCOM1を使います．

さて，JCOM1つまりFFUARTを使用するためには，GP34～GP41を汎用I/Oポートではなく，FFUARTとして機能するように内蔵レジスタを設定します．FFUARTを設定するためには，GPIO Alternate FunctionレジスタのGAFR1_Lレジスタ(アドレス0x40E0005C)のビット4からビット19までを，次のように設定する必要があります．

AF34(GP34)：ビット5/4=‘01’
AF35(GP35)：ビット7/6=‘01’
AF36(GP36)：ビット9/8=‘01’
AF37(GP37)：ビット11/10=‘01’

AF38(GP38)：ビット13/12='01'
AF39(GP39)：ビット15/14='10'
AF40(GP40)：ビット17/16='10'
AF41(GP41)：ビット19/18='10'

さらに，FFUARTのうち，TXDとDTR，RTSの各ピンを，出力方向に設定する必要があります．TXDおよびDTR，RTSの各ピンを出力方向に設定するためには，GPIO Pin DirectionレジスタのGPDR1レジスタ(アドレス0x40E00010)のビット7(DP39)からビット9(DP41)までのビットを1にセットします．

PXA250では，GPDRレジスタはGPIOのピン方向制御レジスタですが，I/O機能としての入出力方向という意味ではなく，GPIOのピンそのものの入出力方向設定なので，そのピンをFFUARTとして使う場合は，FFUARTとしての入出力方向を設定しなければならない点に注意してください．

● RS-232-Cドライバのイネーブル

用途によってはJCOM1を使わずにI/Oポートを多用したい場合も考慮して，JCOM1を使わない場合は，FFUARTとJCOM1の間のRS-232-Cドライバをディセーブルに設定できるようになっています．ここではJCOM1を使ったシリアル通信を行うので，このRS-232-Cドライバをイネーブルに設定しなければ，正常に通信が行えません．

RS-232-Cドライバをイネーブルに設定するには，コンフィグレーションポートのビット9を1にセットします．

● FFUART初期化

次はFFUARTの初期化が必要です．初期化手順は，次のようになります．

(1) ボーレートジェネレータの設定(FFDLL/FFDLH)
(2) データビットおよびストップビットの設定(FFLCR)
(3) 送信および受信FIFOの設定(FFFCR)
(4) モデム制御の設定(FFMCR)
(5) FFUARTのイネーブル設定(FFIER)

PXA250のUARTは，PC/AT互換機などで使われている16550と同じです．ただし，レジスタのアドレスは4バイト単位になっているので，32ビットサイズでアクセスする場合は最下位8ビットのみを使います．

▶ボーレートジェネレータの設定(FFDLL/FFDLH)

まず通信速度の設定が必要です．そのためにはDivisor Latch Lowレジスタ(FFDLL)およびDivisor Latch Highレジスタ(FFDLH)に値を設定します．しかし，このレジスタはすぐにはアクセスできません．Line Controlレジスタ(FFLCR)のビット7を1にすることで，FFDLLレジスタとFFDLHレジスタがアクセスできるようになります．

ボーレートジェネレータに設定する値は，次の公式で表すことができます．

ボーレート値 =14.7456MHz ÷ (16 × DEVISOR)

DEVISORは16ビットの値で，DEVISORのビット7～0にはFFDLLレジスタの下位8ビットが，DEVISORのビット15～8にはFFDLHレジスタの下位8ビットが入ります．

サンプルプログラムでは，通信速度を9600bpsに設定するので，

DEVISOR = 14.7456MHz ÷ 9600bps ÷ 16

で，96となります．よって，FFDLLには96，FFDLHには0を書き込みます．

▶データビットおよびストップビットの設定(FFLCR)

データビットはLine Controlレジスタ(FFLCR)のWLS(ビット1およびビット0)，またストップビットは同じくFFLCRレジスタのSTB(ビット2)で設定できます．ここでは，データビット長を8ビットキャラクタ，ストップビットを1に設定します．よって実際に設定する値は0x3になります．

▶送信および受信FIFOの設定(FFFCR)

送信および受信FIFOを有効にするために，FIFO Controlレジスタ(FIFOCR)のTRFIFOE(ビット0)をセットします．

▶モデム制御の設定(FFMCR)

RTS出力信号をアクティブ(Request to Send)にするために，Modem Controlレジスタ(FFMCR)のRTS(ビット1)をセットします．また，DTR出力信号をアクティブ(Data Terminal Ready)にするために，同じくFFMCRレジスタのDTR(ビット0)をセットします．

▶FFUARTのイネーブル設定(FFIER)

UARTユニットを有効にするために，Interrupt Enableレジスタ(FFIER)のUUM(ビット6) をセットします．

● シリアル送受信

シリアルポートからデータを受信すると，FFLSRレジスタのビット0が1になります．これをメインループで判定します．このビットに1が立ったら，受信データを取り出すためにReceive Bufferレジスタ(FFRBR)を読み出します．受信データは下位8ビットに格納されます．

エコーバックなので，受信したデータをそのまま送信します．データを送信するには，送信バッファに空があるかどうかを確認する必要があります．FFLSRレジスタのビット5が1なら，送信バッファに空きがあります．送信バッファに空きがあることを確認したら，送信データをTransmit Holdingレジスタ(FFTHR)に書き込みます．

なお今回のプログラムは，送信バッファに空きがないと空くまで待つという処理になっていますが，実際にはその間に受信バッファがいっぱいにならないようにフロー制御なども必要です．

ここまで解説したサンプルプログラムを**リスト4**に示します．**リスト4**のプログラムは，データを受信すると7セグメントLEDをカウントアップ表示します．

5 RAMテストプログラム

メモリチェックの方法は，まず，異なるアドレス範囲(範囲長は同じでスタートアドレスのみ異なる)に対して，同じデータを書き込み，それぞれの合計を求めます．それぞれの合計値

〔リスト4〕シリアル通信(エコーバック)プログラム

```
～ 中略 ～
void FFUART_init(int bps)
{
    int i;

    *GAFR0_y = (*GAFR0_y & 0xFFF0000F) | 0x000A9550;/* FFUART使用 */
    *GPDR_y  = (*GPDR_y  & 0xFFFFFC7F) | 0x00000380;/* FFUART(FFTXD,FFDTR,FFRTS出力) */
    *CONFIG |= 0x200;   /* RS-232-Cドライバイネーブル */
    switch(bps){
        case 1200 :
            i=768;
            break;
        case 2400 :
            i=384;
            break;
        case 4800 :
            i=192;
            break;
        case 9600 :
            i=96;
            break;
        case 19200 :
            i=48;
            break;
        case 38400 :
            i=24;
            break;
        default :
            i=96;
            break;
    }
    *FFLCR = 0x80; /* Serial Line Control reg; set Divisor reg access bit */
    *FFDLL = i;    /* Divesor reg; baud tate: i */
    *FFDLH = i>>8;
    *FFLCR = 0x3;  /* Serial Line Control reg; stop:1, data:8bit */
    *FFFCR = 0x1;  /* FIFO Control reg; TandR FIFO enable */
    *FFMCR = 0x3;  /* Modem Control reg; RTS,DTR:1 */
    *FFIER = 0x40; /* Interrupt Enable Register; UART Unit Enable */
}

int main(void)
{
    int led_count;
    unsigned char c;

    /* LEDカウントクリア&LED表示クリア */
    led_count=0;
    *LEDPORT = led_7seg[led_count];

    /* FFUART初期化 */
    FFUART_init(9600);

    while(1){
        if (*FFLSR & 1) {   /* 受信データあり */
            led_count++;    /* LEDカウント+1 */
            if (led_count>15) led_count=0;
            *LEDPORT = led_7seg[led_count];
            c=*FFRBR;       /* データ読み出し */
            /* 送信バッファに空きができるまで待つ */
            while((*FFLSR & 0x20)==0) {}
            *FFTHR=c;       /* データ送信(エコーバック) */
        }
    }
}
～ 以下略 ～
```

を比較することにより，メモリチェックが行えます．

このプログラムでは，0xA0E00000と0xA0E00500から1Kバイト分の範囲でデータを書き込んでいます．次に，それぞれのデータの合計を求め，比較します．合計したデータが一致しない場合は，メモリに何らかの障害が発生していると考えられるので，7セグメントLEDを使ってエラーが発生したことを表示する関数を呼びます．合計データが一致すれば，正常であることを表示する関数を呼ぶというものです(**リスト5**)．

6 CPUモード&キャッシュ設定プログラム

● PXA250の動作モード

PXA250では動作モードとして，Turbo，RUN，IDLE，SLEEPの四つのモードがあります．評価ボードのベースの入力クロックは3.6864MHzで，内部のPLLで逓倍した後，CPUコアやメモリコントローラなど各モジュールに供給されています．

周辺モジュールのクロックは基本的に固定されていますが，CPUコアとメモリクロックは変更が可能で，CCCRレジスタのL，M，Nの各パラメータにより逓倍率が決定されます．

PXA250での設定可能なパラメータと計算式は，次のようになります．

メモリ周波数＝L × 3.6864MHz(L：27，32，36，40，45)

〔リスト5〕RAMテストプログラム

```
void ramCheck(void)
{
    int      i;
    ulong    *memory,*memory2;
    ulong    test_data,start,start2,length;
    ulong sum1=0;
    ulong sum2=0;

    start = SDRAM_CHECKSTART;    /* set value 0xa0e00000 */
    start2 = SDRAM_CHECKSTART2; /* set value 0xa0e00500 */
    length = 0x100;
    memory = (ulong *)start;
    memory2 = (ulong *)start2;
    for( i=1,test_data=0; i<length; i++ ) { /* memory check */
        *memory = test_data;
        *memory2 = test_data;
        memory++;
        memory2++;
        test_data++;
    }

    memory = (ulong *)start;
    memory2 = (ulong *)start2;
    for( i=1; i<length; i++) {   /* memory data sum */
        sum1 += *memory;
        sum2 += *memory2;
        memory++;
        memory2++;
    }

    if(sum1 == sum2) { /* memory data sum compare */
        OK_LED();
    } else {
        ERROR_LED();
    }
}
```

〔リスト6〕CPUモード/キャッシュ変更プログラム

```
    .text
    .global   _ChangeClock
    .global   _EnterTurbo
    .global   _ExitTurbo
    .global   _EnableIcacheBtb
    .equ CCCR_BASE,     0x41300000

_ChangeClock:
    stmfd     sp!, {r1, r2, lr}
    ldr       r1, =CCCR_BASE
    /* CCCR value(L:x27, M:x2, N:x2) */
    ldr       r2, =0x00000241
    /* set CCCR */
    str       r2, [r1]

    /* CCLKCHG value (enter FCS) */
    ldr       r1,=0x2
    mcr       p14, 0, r1, c6, c0, 0
    ldmfd     sp!, {r1, r2, pc}

_EnterTurbo:
    stmfd     sp!, {r1, lr}
    /* CCLKCHG value (enter TURBO) */
    ldr       r1,=0x1
    mcr       p14, 0, r1, c6, c0, 0
    ldmfd     sp!, {r1, pc}

_ExitTurbo:
    stmfd     sp!, {r1, lr}
    /* CCLKCHG value (exit TURBO) */
    ldr       r1, =0x0
    mcr       p14, 0, r1, c6, c0, 0
    ldmfd     sp!, {r1, pc}

_EnableIcacheBtb:
    stmfd     sp!, {r1, lr}
    /* read CP15 Control */
    mrc       p15, 0, r1, c1, c0, 0
    /* set I,Z bit */
    orr       r1, r1, #0x1800
    /* write CP15 Control */
    mcr       p15, 0, r1, c1, c0, 0
    ldmfd     sp!, {r1, pc}

    .align    4
    .end
```

RUNモード周波数＝M×(メモリ周波数)(M：1，2)

Turboモード周波数＝N×(RUNモード周波数)

(N：1，1.5，2，3)

内部クロックは最大で400MHzです．リセット後のCPUのデフォルト値は，

L：27　M：1　N：1

となっており，メモリ周波数，RUNモード周波数，Turboモード周波数のすべてが100MHzとなります．また評価ボードではユーザープログラム実行時は，次のような設定になっています．

キャッシュ　：OFF
モード　　　：RUNモード
内部クロック：100MHz
SDRAMクロック：100MHz

モニタプログラムでM＝2に変更していますが，周波数変更シーケンスは実行していないので，起動後はCPUはRUNモードで100MHzのまま動作しています．

周波数変更およびRUNモードとTurboモードの切り替えは，コプロセッサ14レジスタ6のCCLKCFGレジスタを使用します．レジスタのフォーマットを次に示します．

ビット1：1で周波数変更シーケンスに入る
ビット0：1でTurboモードに入る，0でRUNモード

あらかじめCCCRレジスタに逓倍率をセットしておいてから，CCLKCFGレジスタの設定を行います．CCLKCFGの書き込みには，コプロセッサ関連用の特殊な命令を使用します．

```
MCR p14, 0, Rd, c6, c0, 0
```

● CPUモード/キャッシュ変更プログラム

リスト6に，モード/キャッシュ変更プログラムを示します．これはアセンブラソースですが，次の関数名でC言語から呼べるように記述しています．

▶ void ChangeClock(void)

クロック周波数を設定/変更します．EnterTurbo()を呼び出す前に1度実行します．TurboモードはCPUコアを400MHzで，RUNモードでは200MHzで動作するように設定しています．

▶ void EnterTurbo(void)

ChangeClock()関数はクロック設定を変更するだけで，モードは変更しません．この関数で実際にCPUをTurboモードに切り替えます．

▶ void ExitTurbo(void)

Truboモードの状態から低消費電力動作をしたい場合は，この関数を呼び出すことでRUNモードに切り替えます．

▶ void EnableIcacheBtb(void)

命令キャッシュとブランチダーゲットバッファをイネーブルに設定する関数です．

以上より，評価ボードで最高のパフォーマンスでプログラムを動かすには，

```
ChangeClock();
EnterTurbo();
EnableIcacheBtb();
```

のように，三つの関数を呼べばよいことになります．なお，データキャッシュをイネーブルにするには，XScaleの仕様上，MMUもイネーブルに設定しなければなりません．

*　　　　*

次回は，PXA250に内蔵されているUSBコントローラを使って，簡単なUSBターゲットのファームウェアと，Windows用のサンプルプログラムの作成について解説します．ご期待ください．

参考文献

1) *Intel XScale Microarchitecture for the PXA255 Processor User's Manual*

やまもと・しげひさ　(株)ソフィアシステムズ

IPパケットの隙間から

言論の不自由と
サポート環境の不自由

57

祐安重夫

今世紀に入ってからのアメリカは，本当に信じられなくなった．少なくとも建前としては残っていたはずの「自由」が，「戦争」という言葉をキーワードに，どこかへいってしまったようだ．

WIREDが伝えるところによると，OpenBSDプロジェクトの中心的なプログラマの一人であるカナダ人のセオ・テラードが，今回の「戦争」に関して，プロジェクトがDARPAから助成金を受け取っていることへの「居心地の悪さ」を表明した．この発言に対し，OpenBSDの中心的人物であるペンシルバニア大学のジョナサン・スミス教授から，不快感を伝える電子メールが届き，さらにすぐ後にDARPAからの助成金が打ち切られたことが知らされたというのである．

その後，ペンシルバニア大学には，助成金打ち切りの対象はOpenBSD全体ではなく，テラードが主催するオープンソースプロジェクトに関する集会「ハッカソン」(hackathon)のみであるという通知が届いたという．

DARPAもペンシルバニア大学も，この騒動の背景については口をつぐんでいるが，これが「反戦」の表明に対する報復であることは，誰の目にも明らかだろう．最近では「戦争」に賛成しなかった国々に対して，アメリカが報復を行うかもしれないという動きさえある．

オープンソースソフトウェアも実際の政治や経済と無関係ではいられないことは，これまで表面から論ずることがそれとなく避けられていたが，どうやらそうはいかない時代になってしまったようだ．

経済的側面についていえば，Linuxで商売をしている企業の動向についても，注目していかなければならない．

これまで筆者は，複数のRed Hat Linux 6.2をインストールしたマシンを管理してきたのだが，Red Hatが3月31日に6.2へのサポートを終了してしまった．実際にセキュリティパッチをあてていれば，6.2でも問題なく使用できていたのだが，サポートなしでは必要なパッチを自分でソースを入手し，コンパイルしてアップデートしなければならない．

ちょうど3月31日にCERTから勧告されたsendmailのセキュリティホールなど，どうすればいいのかと思ったら，日本時間では4月1日だが，アメリカ時間ではまだ3月31日の間に，アメリカのRed Hatのサイトでセキュリティパッチが公開され，なんとか対応することができた．しかし，驚いたことに，アメリカ時間で4月1日になったとたん，6.2のサポートページ自体が消滅してしまった．もっとも，その後，同じパッチが日本のRed Hatのサポートページにアップロードされたが．

そこで，複数のドメインに散らばったシステムを，まとめてアップグレードしなければならないことになった．その時点での最新版は8.0だったが，このバージョンも今年の12月31日でサポートが打ち切られることが決まっている．この先，当分サポートが継続されるのは，日本では4月18日にパッケージが発売予定の(つまりその時点ではまだ出ていない)Red Hat Linux 9.0である．

ところで，これまでRed Hat Linux 6.2をサポートしなければならないため，筆者自身も6.2の環境を使用していた．6.2自体は安定したOSであり，サーバとして使用する分には不都合はなかったが，デスクトップ環境としてはしばらく前から，バージョンアップしないと使用できないソフトウェアがいくつかでてきていた．

それに対しては，サポート用のマシンとは別のPCを入手して，それに新しいバージョンをインストールすればいいのだが(すでに以前，Red Hat Linux 7.2のプロフェッショナル版を購入してあった)，ハードウェアの調達が滞っていた．

今回，ともかく新しいPCを調達し，そちらに9.0をインストールした．とりあえずサポート用の6.2は維持しなければならないし，9.0についても評価しなければならない．その9.0だが，サーバ環境としてのテストは現在進行中だが，とくに問題は出ていないようだ．

さて，デスクトップ環境としての9.0はどうだろうか．8.0以降，GNOMEがバージョン2になったのだが，その8.0と比較しても，Nautilusが妙な具合にGNOMEと一体化して，そのデスクトップを停止することができないようだ．ウィンドウマネージャは8.0からmetacityになっており，これまでsawfishを使用してきたので，機能的に不満だし，使い勝手も必ずしもいいとはいえない．

ウィンドウマネージャについては，別のものを選択するGUIがなぜかないので，

```
killall metacity && sawfish &
```

といういささか強引な方法で変更し，ログアウト時に「現在の設定を保存」することでうまくいった．しかしsawfishもGNOME2対応のバージョンになっており，バグがかなりあるようだし，機能的にも必ずしも満足できないが，とりあえずmetacityよりはましである．

サーバ環境については移行できそうだが，デスクトップ環境としてはまだまだ移行に手間がかかりそうだ．

すけやす・しげお インターメディアアクセス

組み込みLinuxをとりまく世界

第1回　組み込みLinuxの長所短所とスマートな導入のための要素

山中　勝

オープンソースの旗手として猛スピードでダッシュを続けるLinuxは，とくにサーバの領域でシェアを伸ばしており，また，デスクトップ分野ではオフィスソフトも充実し，機能的には既存のOSと肩を並べる実力をもつまでになった．また，組み込みシステムにおいてもネットワーク機器，携帯電話などへの搭載も始まり，実用段階に入ろうとしている．

本稿では，組み込みLinuxを採用する上での問題点を検証し，効率良く導入する方法について，現実的な視点から検討する．さまざまなテーマを隔月で6回，解説していく予定である．

なぜ，組み込みLinuxなのか？

インターネットの普及，半導体技術の進歩により，組み込みシステムに要求される機能の増大やシステムの小型化，省電力化などに対応するため，ソフトウェアに占めるコストの増加とTime to Marketに対する開発期間の短縮など，組み込みシステム開発に対する要求の急速な変化に対応することが，開発者に求められている．

この要求に応えるために注目されているのがLinuxである．Linuxは，エンジニアリングワークステーション(EWS)に広く採用されていたUNIXシステムと同等のものであり，組み込みシステムの開発ホストマシンとしても利用されている汎用OSで，ネットワーク，ファイルシステムなど，今日の組み込みシステムに要求される機能を基本的に備えている．

「組み込みシステム」といわれるものは，パソコンの部品のような小さなものから，電話交換機など複数のシステムを統合して制御するものまで，広いスケールをカバーしている．組み込みシステムに搭載されるCPUはパソコンのように標準化されておらず，さまざまなアーキテクチャが採用されている．またOSについては，リアルタイムOS(RTOS)がおもに採用されている．大容量のメモリを必要としないOSとして提供されているものがRTOSであったことなどから，組み込みOSイコールRTOSという認識になってしまった．組み込みOSは，CPUのアーキテクチャ以上に非常に多くの種類が提供されている．

したがって，組み込みシステムを開発する場合はCPU，OS，開発ツールなどについて調査，検討するところから，ネットワーク，ファイルシステムなど必要なコンポーネントの検討にいたるまでの作業を必要とした．これらの選定を行う際，必要な要素の組み合わせが必ずしも要求をすべて満たすわけではなく，また性能のテストも実際に使用してみるまで行えない場合も多く，困難をきわめていた．**表1**に従来の組み込みシステムにおける問題点をあげる．

これらの問題を解決するものとして，オープンソースのシステムはソースコードの提供によりプログラミングの自由が保障されており，ロイヤリティの発生が抑制されている．半導体技術の進歩により高性能なCPU，大容量のメモリが低価格で提供されるようになり，組み込みシステムへの汎用OSの搭載が現実のものとなった．また，オープンソースの発展にともない自由なプログラミング環境がもたらされたことにより，汎用システムにおけるオープンソースへの移行が，組み込みシステムにおいても主流になりつつある．オープンソースのOSとしてはLinux以外にもFreeBSDなど多くのシステムがあるが，Linuxの自由で現実的な方針による進化のスピードが今日の繁栄を支えており，変化の激しい組み込みシステムにおいてもこの成果を享受できる．**表2**に組み込みシステムにおけるLinuxの長所

〔表1〕従来の組み込みシステムにおける問題点

●CPU，OS，開発ツール，コンポーネントの組み合わせに制限があり，望みのシステムが構成できない
●OSベンダからソースコードの提供が行われない，あるいは高額な費用が必要
●開発システム変更のたびにコードの修正，ツールの学習が必要
●フラグメント化されたシステムのため，バグフィックスのスピードが遅い

〔表2〕組み込みシステムにおけるLinuxの長所

●GPLの下でソースコードが提供されており，プログラミングの自由が保障されている
●開発ツールは多くのCPUをサポートしているGNUツールで統一されている
●世界中のプログラマが日々システムの開発，テストを行っており，新しい機能の導入やバグフィックスのスピードが非常に速い
●Linuxホスト上で開発されたソフトウェアがターゲット上で利用可能
●ロイヤリティフリーである

をあげる．

今日の複雑で多様な組み込みシステムを構築する上で，ネットワークをはじめとする多くの要求を満たし，負担の増加したソフトウェア開発の効率化にはLinuxが非常に適しており，大小さまざまな規模のシステムにおいて採用される要因となっている．

エンジニアのひとり言：その1

GPLを悪者呼ばわりするのであれば使わなければいいのに．オープンソースはもうからないけれど強力なからくりだ．

組み込みLinux導入の問題点

組み込みシステムにおける多くのメリットをもたらすLinuxではあるが，いくつかの問題点もかかえている．組み込みシステムにおけるLinuxの問題点を**表3**にあげる．

〔**表3**〕**組み込みシステムにおけるLinuxの問題点**

●オープンソースは無保証である
●ソースコードは公開されているが，簡単に理解できるわけではない
●GPLはプログラミングの自由を保障するライセンスで，修正した内容は公開しなければならない
●リアルタイム機能がサポートされていない場合が多い

オープンソースで提供されるソフトウェアはいっさいの保障が行われないため，利用者の責任において使用することとなる．実際に提供されるソフトウェアはテストの有無，コードの品質を含めバラつきがある．したがって，製品に搭載する場合は商用OSのように安定した品質でテスト済みのものが提供されないため，自前でテストを行うか専門ベンダによってテスト済みのものを利用することとなる．「オープンソースはフリーである」という意味を「タダ」と解釈している場合が多く見られるが，本来は「プログラミングの自由」が保障されているという意味なのである．

また，ソースコードは公開されているが，汎用OSであるLinuxのソースコードはC言語の教科書に記載されているような簡単なものではない．ビルドの際に用いられている複雑なMakefileやマクロ定義を解読するためには高いスキルを必要とする．

したがって，商用OSのようにテスト済みのものを提供するベンダのサポートサービスを利用し，安心してアプリケーション開発に集中するのがスマートなLinuxの利用方法の一つとなる．

Linuxを利用する上で，ライセンスの問題が採用を躊躇させる要因となる場合がある．ご存知のようにLinuxはFSF(Free Software Foundation)のGNU GPLにて提供されている．GPLはプログラミングの自由を保障するため，修正を加えたカーネルなどのソースコードを公開する義務を課している．アプリケーションに関してはLGPLと呼ばれる，GPLよりゆるいライセンスによって，ダイナミックリンク形式でLinuxのライブラリとリンクすることにより，実質的にソースコードの公開義務を免除している．これ以外にもC++のライブラリのように，GPLではあるものの例外事項によりソースコードの公開が不要なものも存在する．これらの例外は，ほかのオープンソースソフトウェアとの親和性やFSFの戦略的な要因から設けられている．

また，Linuxのデバイスドライバにおける GPLの扱いについては，著作権者であるLinus Tovaldsの追記事項の解釈により，ローダブルモジュール形式を用いればソースコードの公開は必要ないと解釈される場合があるが，厳密なFSFの解釈にしたがって公開の義務があると解釈される場合もある．利用者の現実的な立場から見れば，セキュリティなど公開することにより情報の安全性などに影響を与える可能性のあるもの，商用コンポーネントの利用の際に，GPLとのライセンス上の矛盾が発生するものなど，ソースコードの公開が行えないものとの協調利用が必要なものがある．

これらのライセンスについては，法律家においても100%共通の解釈が行われるわけではなく，実際に訴訟が起こってみないと解答は得られないかもしれないが，ソースコードの実装の方法により限りなくクリアにすることも可能である．

基本的にはソースコードを公開する精神がたいせつであり，公開することにより公に認められ，製品ベンダに対する信頼に結びつく．実際にソースコードを公開したことによりヒットした拡張ボードなどの事例もある．

オープンソースのライセンスに関する情報については，雑誌などの断片的な情報によりGPLは危険なものという誤った解釈をされている場合も多く，正しく認識してGPLにともなうリスクを把握することにより，Linuxが充分製品に利用可能であることを認識していただけるものと思う．オープンソースに関する書籍や，次に示すWebサイトの情報をぜひご一読いただきたい．

http://www.opensource.org/

Open Source Initiative(OSI)

エンジニアのひとり言：その2

展示会のデモプログラムをまずPCのLinuxホスト上で作成して，完成後ターゲット用のクロスツールでリビルドしたところ，何事もなかったように動作した．アプリの開発は驚くほど簡単である．一度お試しあれ．

組み込みシステムにおいて必ず問われるのがリアルタイム機能である．一般的に，割り込みやイベントに対する応答時間を最悪値として保障する「ハードリアルタイム」と，確率的に保証する「ソフトリアルタイム」に分類される(**図1**)．ハードリアル

タイムは，車両，FAシステム，原発などのように，いかなる場合でも絶対に応答時間が保証されないと災害や甚大な損害を生じるシステムに採用されている．ソフトリアルタイムは，応答時間がある確率以上で保証されていればよいシステムで利用されている．従来の組み込みシステムにコンパクトなOSとして採用されてきたRTOSだが，本当にハードリアルタイム機能が必要な場合は，組み込みシステム全体においても非常に少ない分野に限定されており，本質的に一般的なLinuxで充分対応可能となっている．

汎用システムであるLinuxは，システムコールなどのカーネル内部の処理実行中は割り込みを禁止状態にするため，割り込みに対する応答時間にバラつきがあり，リアルタイムシステムに必要とされる割り込み応答時間の保証ができない．また，仕事の処理単位であるプロセスに対して平等にCPU時間を割り当てるよう，動的に優先順位を割り当てるポリシが採用されており，割り込みに対応したプロセスの応答時間も保証できない．これに対しリアルタイムシステムでは，カーネル内部の処理実行中においても，可能なかぎり割り込みを有効にしており，割り込み応答時間を保証している．また，仕事の処理単位であるタスクやスレッドに対して静的な優先順を割り当て，イベントなどに対するタスクやスレッドの応答性を確保している．しかし「リアルタイム」という名称から，リアルタイムシステムのほうが処理速度も速いと思われている傾向がある．実際は割り込みを受け付ける時間が長い分，割り込みによる実行中の処理の中断によるコンテキストのスイッチが頻繁に発生し，非リアルタイムシステムに比べて，全体のパフォーマンスは割り込みの頻度に比例して低下する．

> エンジニアのひとり言：その3
>
> **どんなにすぐれたRTOSでも間に合わない場合がある．そんなときは割り込みベクタから直接処理を行うことになるわけで，ちょっとしたリアルタイム機能なら，大げさなシステムがなくても簡単に対応できる．**

Linuxにおいても，さまざまなリアルタイムシステムが提供されている．カーネル内部の処理部分に割り込み禁止状態を可能なかぎり取り除くコードを追加したもの，プロセスのスケジューリングをリアルタイムシステムに変更したもの，あるいはほかのリアルタイムシステムとのハイブリッドシステムなど，多彩である．

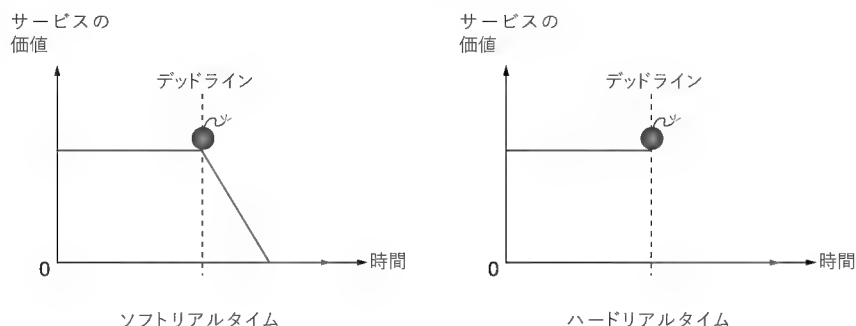
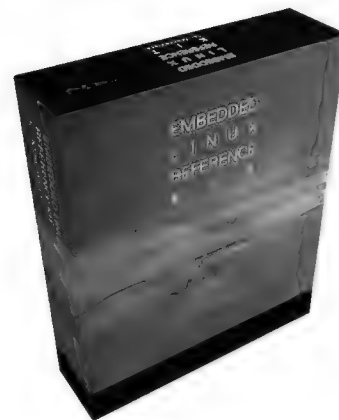
組み込みLinuxのスマートな導入

従来から使用しているRTOSからLinuxへの移行は単純ではない．多くのRTOSは，タスクあるいはスレッドと呼ばれるメモリ管理などを含まない比較的シンプルなジョブコントロールで，Linuxのプロセスのように仮想記憶をサポートする汎用システムとは異なる．また，ファイルシステムやネットワークシステムも標準的にサポートしており，これらの機能をオプションのコンポーネントのタスクとして実装しているRTOSとは異なり，複雑なシステムである．

また，ソースコードは公開されているもののLinuxのコードは複雑で，一夜にして理解するのは不可能である．したがって，短期間でLinuxシステムを構築するためには，Linuxに精通したエンジニアを確保して自社の製品へのポーティングを行うか，専門の組み込みLinuxベンダに開発を依頼することになる．いずれにしても，人的あるいは多大な開発コストが必要となる．

さらにLinuxは無保証のため，製品レベルでの使用に耐えるようにするためには，開発ツールを含め厳密なテストが必要である．自前でこれらのテスト体制を用意するのはLinuxのカーネル，開発ツールなど複数のエキスパートを確保する必要がある．また，これを実現するためにはかなりの費用が必要となり，組み込みLinuxベンダが提供するサポートサービスは一見高価な感じがする．しかし，同等のことを自前で行うことを考慮すれば充分リーズナブルであり，実際の製品開発においては緊急事態にも迅速なサポートが受けられる安心を得るための費用として納得できるものである．

小規模なプロジェクトやプロジェクトの初期におけるプロトタイプを構築する場合などの限られた開発コストの下でLinux

〔図1〕ハードリアルタイムとソフトリアルタイム

〔写真1〕組込みLinux評価キット

〔表4〕「組込みLinux評価キット」の概要

製品内容
● CD：MontaVista Linux，LSP，GNU Tools，ブートローダ（x86を除く），ドキュメント ● 評価キット利用ガイド ● 評価キットボードガイド ● ユーザー登録用紙
サポートCPU（一部開発中）
● x86，ARM，XScale（Strong ARM），SH

〔図2〕組込みLinux導入パス/サポート体制

を導入する方法としては，Linuxが移植されているターゲットボードを利用してスタートするのが近道である．また，トレーニングやセミナなどの教育プログラムを受講してLinuxの正しい利用方法を短期間で習得するのも有効な方法といえる．

これらの方法を状況に応じて利用することによりLinuxの導入をステップバイステップで，Linuxに関するノウハウを蓄積しながらスキルを高めて「Linuxの達人」を養成することにより，組み込みLinuxの能力を有効に引き出すことができる．

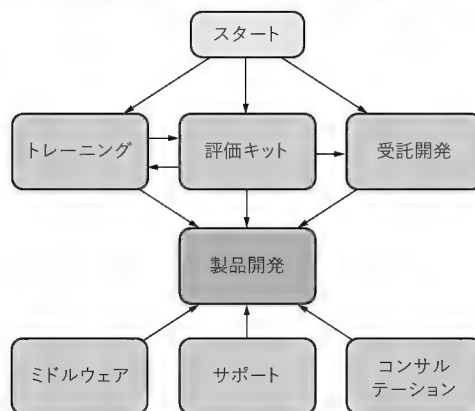
さらに実際の製品を構成する要素としては，GUIシステム，メモリカードシステムなど基本的なLinuxではサポートされていないコンポーネントの調達も必要となる．これらについては，サードベンダにより提供されている製品を有効に活用することにより，開発スピードをかせぐことが可能である．

組み込みLinuxのスマートな導入方法としては，次に示すようにステップバイステップで組み込みLinuxのスキルを身に付けるとともに，専門ベンダが提供するテスト済みのLinux，オプションのコンポーネントを利用することにより信頼性の高いシステムを迅速に構築することである．これによりコスト，開発スピードのバランスを取りながら効率をよくLinuxをスマートに導入することが可能となる．また，製品の構成や実装方法に関する問題解決の際には，専門家によるコンサルテーションやサポートも必要となる．

> エンジニアのひとり言：その4
>
> **Linuxのサポート費用が高いとボヤく前に，複雑なシステムや開発ツールに精通したエンジニアを探すだけでも容易ではなく，多くの要素をもつサポート体制を構築するだけでもたいへんなこと思ったら，お買い得に見えてくる．**

「組込みLinux評価キット」の利用

実際にスマートなLinuxの導入を行う場合，プロトタイプに用いる廉価なLinux搭載のターゲットボード，教育プログラム，受託開発やサポート，コンサルテーションなどの支援プログラムを統合的に提供可能なベンダは限られてしまう．

今回，(株)イーエルティより発売された「組込みLinux評価キット」(**写真1**)は，組み込みLinuxとして定評があり使用実績も豊富なMontaVista Linux Professional Editionをベースとして，国内で入手の容易なターゲットボードのボード依存部分であるLSP(Linux SupportPackage)とLinuxのブートに必要なブートローダをポーティングして，購入後すぐに利用できるパッケージとして提供されており，Linux導入の入り口としての要素を備えている．**表4**に「組込みLinux評価キット」の概要を示す．

「組込みLinux評価キット」による評価が終了後の製品開発におけるサポートについては，強力なパートナシップをもつモンタビスタジャパンが提供する優秀なサポートを受けることが可能で，評価システムと同じ組み込みLinuxを引き続き利用できる．

また，「ToDo(トド)」と称されるアプリケーション開発に必要な日本語変換システムやブラウザなどのコンポーネントをリーズナブルな費用で評価可能なソリューションが提供されており，製品開発に必要な要素を統合して調達可能となっている．さらに，エンジニアのスキル向上のため，「組込みLinux速習コース」や「ブートローダポーティングコース」などの教育プログラムも利用可能となっている(**図2**)．そのほか，Linux開発環境を含めたJTAGプローブや，IDEなどのLinux開発環境の提供も行っている．

いずれも，組み込みLinuxをスマートに導入する上で必要となるソリューションをシームレスに提供することにより，組み込みLinuxを搭載した製品開発を容易に行えるソリューションを提供している．詳細は，以下のWebサイトを参照いただきたい．

http://www.emblit.co.jp/

次回は，「組込みLinux評価キット」の詳細などを解説する予定である．

やまなか・まさる　(株)イーエルティ

ファイアウォール，帯域管理装置などのプラットホームとなる

アプライアンス機器「InterWay/GB」の概要

青木　弘／新井雅樹／遠藤俊也／鈴木明彦／中井清元

はじめに

「アプライアンス機器」とは，特定機能に特化したハードウェア/ソフトウェア一体型の専用装置のことである．専用装置化により，操作が簡単で，導入期間の短縮，使いやすさなどを実現している．筆者の会社〔(株)PFU〕では，負荷分散装置，帯域管理装置，ファイアウォール，ウイルスウォールなどの各種アプライアンス製品を開発/提供している．また，これら製品で実績のあるハードウェアを，ソフトウェアベンダ，システムインテグレータ，ネットワークインテグレータなど向けにアプライアンス用ハードウェアプラットホームとして提供している．

1　製品の概要と特徴

1.1　概要

InterWay/GBは，IA(Intel Architecture)ベースのラックマウント(1U)の筐体にギガビットEthernetのポートを4ポート標準装備している．マザーボードから筐体まで自社開発を行い，表示/操作機能の装備，24時間365日連続運転を前提とした高信頼性の実現，長期提供の実現，IDC(Internet Data Center)用途を配慮した前面ケーブリングの採用など，高性能アプライアンス構築に最適なハードウェアプラットホームをめざして開発した．また，InterWay/GBへのアプリケーションソフトの実装を補助するLinuxベースの開発キットも提供している．**図1**に外観を示す．

1.2　製品の特徴

図2に内部レイアウト図，**図3**に前面/背面図を示し，特徴を説明する．

〔図1〕InterWay/GBの外観

● Gigabit Ethernet × 4ポートをマザーボードに標準搭載

1）高速で柔軟性に富むネットワーク接続

Gigabit Ethernetコントローラ「Intel 82544GC」4個をマザーボードに搭載し，独立したMACアドレスを割り付けた4系統の10Base-T/100Base-TX/1000Base-TのLANインターフェースを標準装備した．

2）処理性能

ギガビットインターフェースに応じた処理性能を実現するため，プロセッサにPentium III 1.26GHz(2次キャッシュ512Kバイト)，システムチップセットにServerWorks製「HE-SL」を採用した．

3）高速メモリアクセス

メインメモリは，PC133の512MバイトDIMM2枚(ECC付き)を実装し，合計1Gバイトである．最大メモリ転送レートは，2枚のDIMMに対しインタリーブアクセスを行うため，PC133メモリの2倍となる2.1Gバイト/秒である．

4）高速データ転送と広いバス帯域

PCIバスを3系統装備している．2系統は64ビット/66MHz PCIバスで高速なデータ転送を実現している．64ビット/66MHz PCIバスのうち1系統にはギガビットEthernetコントローラを接続しており，もう1系統は拡張PCIスロット用である．残る1系統は32ビット/33MHz PCIバスで，IDEインターフェース，レガシーデバイスなどを接続している．

● システムの適用範囲を広げる64ビット/66MHz PCI拡張スロットを装備

64ビット/66MHz PCI拡張スロットを2スロット装備している．ファイバチャネルを実装し，ネットワークストレージアプライアンスへの展開やSSL(Secure Sockets Layer)アクセラレータカードを装備するなど，ユーザーのアプライアンス構築企画に柔軟に対応できる．

● 24時間連続運転・長期安定供給を前提とした設計

部品採用においては，Embedded Intel Architecture Processorの採用，UNIXサーバ用電源ユニットの採用など，長寿命/高信頼部品を選択し，さらに，厳密な冷却設計により，24時間×365日×5年の連続運転を可能する信頼性と装置寿命

を確保した．

1）アプライアンス向け高信頼性機能を装備

1チップマイコン，センサおよびCPLD(Complex Programmable Logic Device)で構成した外部レジスタからなるRAS機能をオンボードで実装している．

● オンボードの監視機能により異常を検知しLED，液晶パネルなどで通知

電源監視，CPUファン/装置ファンの監視，CPU温度監視，吸気温度監視機能をオンボードに実装している．異常を検出すると警告LEDを点燈，割り込みを発生し，OSが動作可能状態であれば異常処理を起動できる．また装置前面の液晶パネルに異常内容を表示する処理を実装し，迅速なトラブル対応を実現できる．ファン異常検出の場合にOSが動作不可状態だと，一定時間後にハードウェアで電源を切断する．

● ウォッチドッグタイマ機能によりシステムハングアップも素早く検出可能

0.25秒から32秒の間で可変のウォッチドッグタイマを装備している．本機能を使用するとシステム異常をすばやく検出可能で，二重化システムでの切り替えを高速化できる．ウォッチドッグタイマで異常を検出したときは，割り込みを発生させOSの異常処理を起動する．OSが動作不能の場合は1秒後にハードウェアにより再起動または電源オフを行う．再起動か電源オフかは設定可能である．

2）24時間連続運転を支える冷却設計

1Uサイズの薄型筐体，24時間連続運転，装置寿命5年を前提としたネットワーク装置の条件下で，信頼性と装置寿命を確保するには，厳密な冷却設計を行うことが重要で欠かせない．

本装置は，熱と空気の流れをシミュレーションする3次元熱流体解析を駆使し，筐体レイアウト，マザーボード上の部品配置などを検証し，設計へのフィードバックを実施した．

● システム冷却

本装置は前面に表示パネルやI/Oユニット，コネクタを集中的に実装するため，前面の吸気面積を十分に確保することが困難なレイアウトとなった．さらに前面から吸気し後面へ排気するラックマウント装置の制限があり，上面/側面からの吸気は行わないものとした．そのためファンのサイズは筐体高さの制約から40mm角とし，吸引力を強化するため静圧が大きいファンを選定した．ファンの個数は，装置内部の空気温度上昇を15℃以下におさえるため，電源ユニットの冷却に1個，マザーボード部品の冷却に1個，3.5インチディスク用に1個と合計3個の軸流ファンを実装し，装置内で発生する約100Wの熱を排出させている．

● CPU/システムチップセットの冷却

装置内で発生する熱の約40%はCPUとチップセットが占めている．その中でもっとも発熱密度が高いCPUは，最大約30W発熱する．これを冷却するため，局所冷却用CPUクーラを使って冷却している．このCPUクーラは軸流ファンとヒートシンクを組み合わせたもので，装置高さ制約より薄いものを選定する必要があった．しかし薄いものは一般的に放熱性能が劣る．そこでアルミと銅フィンを組み合わせたハイブリッドヒートシンクを採用することで，冷却性能を向上させた．さらにCPUクーラから排気された熱風は，システムファンによりすみやかに装置外へ排熱される構造を採った．

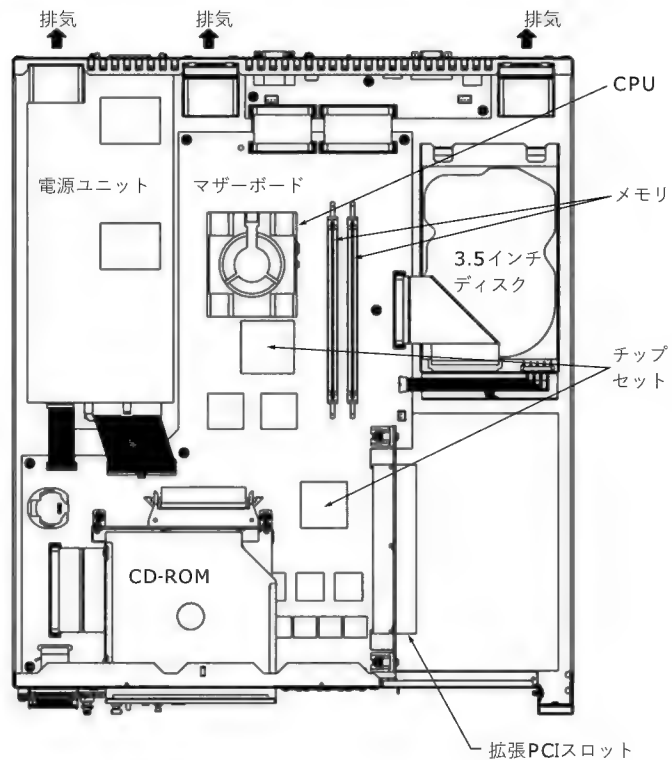
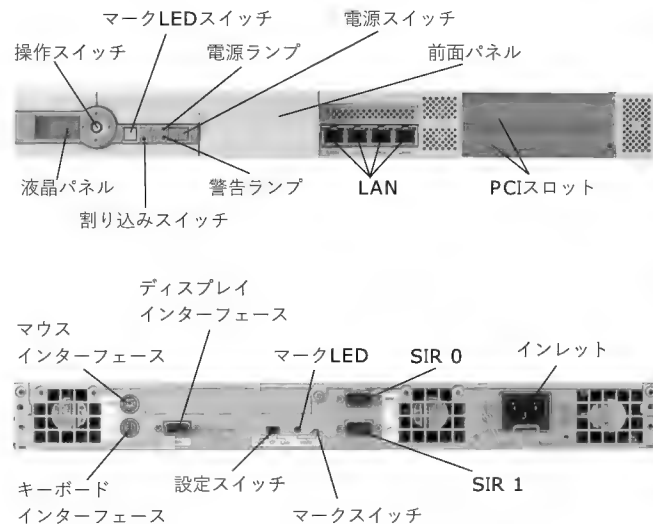



図4にシミュレーション用の解析モデルとCPU断面の温度分布，風速分布を示す．次にシステムチップセットの冷却だが，

〔図2〕内部レイアウト

〔図3〕前面/背面図

チップセットも高速化にともない数年前のCPU並みに発熱する．この冷却にはピン型ヒートシンクを採用し，高い放熱性能を得るためフィン高さを18mm，25mmの2種類用意し，発熱量の大小によりフィン高さを決めた．

これらの冷却部材により各部品の動作保証温度を満たし，24時間365日連続運転を前提とした冷却性能を実現している．また冷却性能だけでなく，冷却コスト/組み立て性/保守性にも十分考慮したものとなった．

● **アプライアンス向け簡単操作機能などの装備**

1）表示/操作機能

装置前面に8桁×2行の液晶パネル，5方向の操作スイッチがあり，簡単なシステムメニューの表示や操作ならば装置単体

〔図4〕熱解析結果例

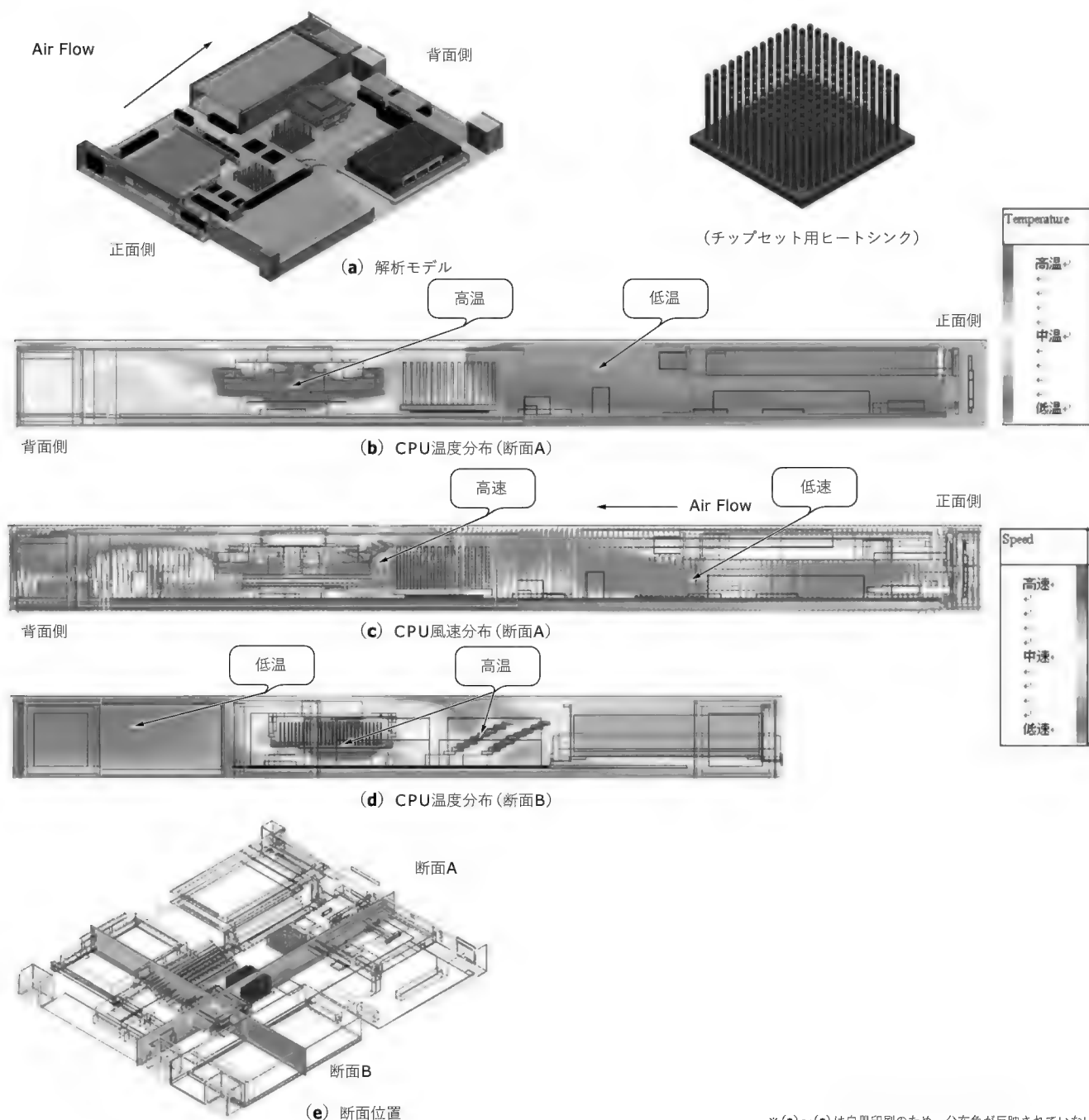
（チップセット用ヒートシンク）

（**a**）解析モデル

（**b**）CPU温度分布（断面A）

（**c**）CPU風速分布（断面A）

（**d**）CPU温度分布（断面B）

（**e**）断面位置

※（**a**）〜（**e**）は白黒印刷のため，分布色が反映されていない

で行える．また，ソフトウェアからの制御も可能な2色の電源LED，警告LEDおよび割り込みを発生させてメンテナンス処理などを起動可能な割り込みスイッチも装備しており，ディスプレイ/キーボード/マウスレスシステムの構築など，簡単運用の実現を支援するハードウェア機能を備えている．

2）コンパクトフラッシュスロットの装備

装置前面パネル内に，Type Iのコンパクトフラッシュスロットを装備している．これにより，たとえば現地インストール時の個別設定情報提供などに，コンパクトフラッシュカードを利用できる．また，コンパクトフラッシュからのブートも選択可能であり，ディスクレスシステムの実現も可能である．

3）各種インターフェースの装備

装置背面にシリアルポートを2ポート装備し，コンソール装置の接続とUPS装置の接続などに利用できる．また，アプライアンス装置の供給者が行うインストール作業，トラブル調査など向けに，ディスプレイ/キーボード/マウスインターフェースも装備している．

● **IDCでの設置に配慮した設計**

LANポートは装置前面に配備し，スイッチなどのネットワーク機器との接続性を向上させている．また，装置の前面背面で連動したMark LED付きスイッチを装備し，19インチラックに多くの装置が高密度で搭載されている状態でも，操作対象とする装置を簡単に識別できる．さらに，奥行きの短いネットワーク機器向けラックでの設置を考慮し，1Uの薄型サイズで装置奥行498mmの小型化を実現している．

● **ユーザーアプリケーションソフトの実装に使用するLinuxベースの開発キットを提供**

InterWay/GBでは，アプリケーションソフトを組み込む際のソフトウェア開発に使用するInterWay/GB開発キットを提供している．これは，InterWay/GBの本体装置，ハードウェアマニュアルとともに，InterWay/GBの独自機能を制御するためのサンプルソフトウェアと開発を支援するサポートを提供する．

サンプルソフトウェアCD-ROMには，液晶パネル，操作スイッチ，RAS機能などInterWay/GBの固有ハードウェア制御機能を使用するためのドライバ，コマンド，ライブラリ（ソースコード，ドキュメントを含む）を収録している．また，インストール用CD-ROMを作成する際のInterWay/GB固有機能の修正/追加に関するヒントなども含んでいる．これらのサンプルソフトを使った機能実装例を，以下に示す．

1）固有ハードウェア制御機能を利用した，ハードウェア異常の検出，表示機能
- 温度異常，ファン異常を検出時にAlarmLEDを点燈し，液晶パネルに異常内容を表示

2）液晶パネルと操作スイッチを使用した簡易メニュー表示や簡易設定機能
- 装置動作（システム停止，再起動など）の指示
- IPアドレスやネットマスクなどの設定
- 現在の動作状態（ネットワーク接続状態，負荷など）の表示

なお，本開発キットに添付されるサンプルソフトウェアの使用条件は，ソフトウェアライセンス規約であるGPL（GNU General Public License）にしたがう．

おわりに

InterWay/GBは，ハードウェアプラットホームとして提供しており，ハードウェアの外観色，ロゴなどのカスタマイズについては商談ごとに相談のうえ対応できる．また，各種ネットワークアプライアンス開発の実績に基づいた，以下のLinux関連技術についても，InterWay/GBに付随して対応可能である．

1）Linux高信頼性技術
- 二重化によるフェイルオーバシステムの構築

2）Linux組込み技術
- コンパクトフラッシュを利用したディスクレスシステムの構築
- ディスプレイ，キーボード，マウスレスシステムの構築

以上のように，InterWay/GBは高性能アプライアンス装置向けに適合したハードウェアプラットホームとして実現した．今後はより広い業種に採用されるよう，機能の拡充をはかっていく．

あおき・ひろし/あらい・まさき/えんどう・としや/すずき・あきひこ/なかい・きよもと　（株）PFU

第20回

オブジェクト指向を採用したスクリプト言語 —— Scriptol

水野貴明

今回紹介するScriptolは，オブジェクト指向を採用したスクリプト言語である．この言語は，実行用のバイナリファイルを作成できるほか，インタプリタを利用して，Webページに埋め込んで実行することもできる．とはいっても Scriptol 自体にインタプリタは実装されておらず，バイナリファイルを作成する機能もない．

Scriptolに搭載されている「コンパイラ」は，Scriptol のプログラムを，C++もしくはPHPのプログラムに変換するものである．

DATA

名称：Scriptol
作者：D.G. Sureau 氏
Webサイト：`http://www.scriptol.com/`
対象OS：Windows，Linux
現在のバージョン：3.4（2003年4月現在）

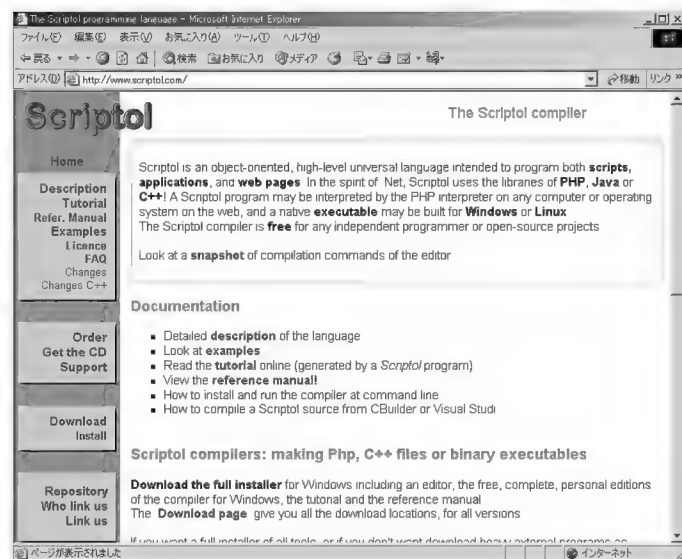
インストールと作業環境

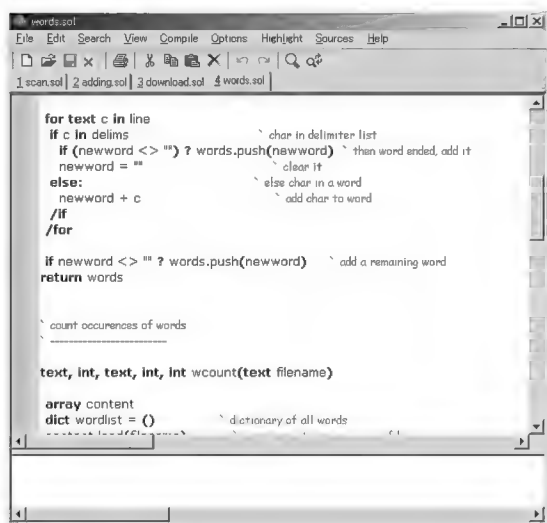
Scriptolは，ソースコードが公開されておらず，Windows版とLinux版のコンパイル済みのバイナリが配布されている（**図1**）．今回は，おもにWindows版を利用して検証を行った．

Windows版はインストーラ形式になっており，指示にしたがって簡単にインストールを行うことができる．今回は，Windows 2000が動いているマシンへのインストールを行ったが，とくに問題は発生しなかった．なお，Linux版はtarballで配布されており，それを展開すれば必要なものは中にすべてそろっている．

さて，Windows版もLinux版も基本的な機能はまったく同じで，インストールされたファイルの構成もほぼ同じである．ディレクトリの中を見てみると，Scriptolの本体であるsolc（C++へのコンバータ），solp（PHPへのコンバータ），サンプルプログラム，プログラムのビルド時に利用するライブラリなど，いくつかのファイルが置かれている（**図2**）．

また，Windows版には，IDE（統合開発環境）が付属している．このIDEは，見た目はシンプルなものだが，キーワードのハイライト機能もきちんと備えており，スクリプトの作成から，C++，PHPへのコンパイル，ビルドや実行まで行うこと

〔図1〕**Scriptolの配布元**（`http://www.scriptol.com/`）

〔図2〕**Windows版のScriptolのファイル群**（コンバータやマニュアル，サンプルプログラムなどがまとめて入っている）

〔図3〕Windows版のIDE

ができる(**図3**).

このIDEのエディタは，Scriptolだけでなく，C++，HTMLといったファイルのハイライト表示にも対応しているため，C++やPHPに変換し終わったプログラムについても，表示/編集を行うことができる.

ちなみに，Linux版のIDEは公開されていないが，エディタの設定ファイルを見ると，Windows向けの記述のほかに，Linux向けの記述も見られるので，Linux版も今後公開されるようだ.

言語仕様

では続いて，Scriptolの言語仕様について見ていくことにする．Scriptolのサイトには「7 rules for a programming language」というドキュメント[注1]が掲載されており，そこに言語をデザインするにあたって作者が定めた七つのルールが書かれているので，まずはそれを見ていただきたい．その七つとは，次のようなものである.

1. 人間の思考をそのままプログラミングできること
2. 安全であること
3. 他の言語との共通点をもつこと
4. 客観性をもつこと
5. 片寄った言語にしないこと
6. 移植性が高いこと
7. 学ぶのが簡単であること

このドキュメントには，それぞれのルールの詳しい意味についても述べられているが，その内容についてはここでは省略する．これらのルールからも，Scriptolが「これまでできなかったことを実現する」というよりは，「なるべくわかりやすく，簡単に物事を行う」ということに重点を置いた言語であることがわかる.

注1：http://www.scriptol.org/seven.php

〔リスト1〕簡単なサンプル

```
text factorize(int x)
  int d,q
  text ans = x.toText() + " = "
  while x>=4 and x mod 2=0
    ans + " 2 * "
    x / 2
  /while
  d = 3
  q = x / d
  while q >= d
      if x mod d=0
        ans + d.toText() + " * "
        x = q
      else
        d + 2
      /if
  /while let q = x / d
  ans + x.toText()
return ans


print factorize(33201)
```

さて，実際の言語仕様は，PerlやPHP，Lua，BASICといったさまざまな言語の特徴が取り入れられているという印象を受ける．他の言語と共通点を多くもつことについては，前述した第3のルールでも示されているが，これまで他の言語を使っていたプログラマが無理なくScriptolを理解できるようにするために，非常に重要視されているようだ.

まず，簡単なScriptolのサンプルプログラムを**リスト1**に示す．BASICのように，各構文の終わりは改行で表され，セミコロンなどで明示的に終了させる必要はない点や，制御構文の終わりが「`/for`」や「`/if`」といった形で表されるなどの特徴が見て取れる.

● 変数

Scriptolに存在する変数型は`text`(文字列)，`boolean`(論理値)，`int`(整数)，`real`(実数)，`number`(数値すべて)といった一般的な変数型のほかに，配列を扱う`array`型，連想配列を扱う`dict`型，そしてどんなデータでも扱うことができる`dyn`型，ファイルを扱うファイル型などの変数型が用意されている.

Scriptolでは，変数をはじめにきちんと定義する必要があり，変数型の指定もその際にきちんと行わなければならない．変数の定義は，C/C++の定義文と似ており，変数型の後に変数名を記述する．また，次のように定義文で変数の初期化を行うこともできる.

```
int  i = 10
```

さらに，動的な型変換は行われず，たとえば数値を文字列変数に変換するには，次のように書かなければならない.

```
text t = i.ToText()
```

このあたりの仕様は，変数型の指定を行わず，動的な型変換をもつのがあたりまえとなっている既存のスクリプト言語とは大きく異なっているが，ScriptolがC++への変換を行うものであることを考えると，仕方がないことなのかもしれない.

また，上記の「`i.ToText()`」という書き方からもわかるように，各変数型は実際にはメソッドを備えたクラスとなっており，

数値データであれば，toText(テキストに変換)，toInt(整数に変換)といったメソッド，文字データであれば，length(文字列長を取得)，trim(両側のスペースを取り除く)，find(文字列検索)といった，他の言語でもおなじみのメソッドが用意されている．

演算処理の書き方は一般的な言語とほぼ同様だが，たとえば，変数aの値と変数bの値を加算して変数aに格納する，といった場合，次のように記述することができる．

```
a+b
```

つまり，演算結果の保存先を明示的に指定しなかった場合，最初に指定した変数に結果が格納されるのである．これは，三つ以上の変数を指定した場合も同様である．次の場合にも，演算結果は変数aに格納される．

```
a+b+1
```

ただし，いちばん左に数値がくるような記述は構文エラーとなる．

```
1+a+b
```

● 配列/連想配列

続いて，配列と連想配列についてである．配列，連想配列では，どちらも要素は[ ](大括弧)付きで指定する．

```
array a
dict  d
a[10] = 100
dict["key"] = "TEXT"
```

配列の要素の範囲は動的に決められ，あらかじめ指定する必要はない．また，各要素はすべてdyn型(どんなものでも格納できる変数型)になるため，同じ配列の中に数値と文字列を混在させることも可能である．また，配列/連想配列にはイタレータの機能があり，begin(ポインタを先頭の要素にセット)，end(ポインタを最後の要素にセット)，inc(ポインタを次の要素に移動)，dec(ポインタを前の要素に移動)といったメソッドを利用して，データを順に操作していくことができる．

```
a.begin()
while a[] <> nil
  print a[]
  a.inc()
/while
```

さらに配列は，push(いちばん最後に要素を追加)，pop(最後の要素を取り出す)，unshift(配列の先頭に要素を追加)，shift(配列の先頭から要素を取り出す)といったメソッドを利用し，スタックやキューとして利用することもできる．

● 制御構文

Scriptolの制御構文は，比較的充実している．if文，for文，while文，do文などは，BASICやC/C++，Perlなどの言語でもおなじみのものである．そのほかにユニークなものとしては，while letブロックがある．これは，whileブロックの最後に演算処理を追加するものである．

Scriptolによるループ処理は，break(ループを抜ける)，continue(その後の処理をスキップし，新しくループ開始点から処理を開始する)による流れの制御が行える．これは，他の言語にもある機能だが，ループ処理の最後でカウンタ変数をインクリメントするような場合，continueを使ったことで，そのインクリメント処理までもがスキップされてしまい，無限ループに突入するといったことが起こる可能性がある．

while letブロックは，以下のようにwhileで始まり，letで終わるブロックで，let文に続いて演算処理を記述することで，continueでループがスキップされた場合でも，let文に書かれた処理だけは実行されるようにするというものだ．

```
int x = 0
while x < 20
  if (x mod 2) = 0
    continue
  /if
  print x
let x + 1
```

もう一つ，Scriptolの特徴的な制御構文としてscanがあげられる．これは，配列/連想配列の要素，それぞれについて処理を行いたい場合などに利用する．たとえば，以下のコードは，配列aの各要素を10倍する．

```
scan a
  a[] * 10
/scan
```

また，scanでは，複数の配列/連想配列を同時に扱うこともできる．その場合は，それらの中でもっとも要素数の少ないものの回数分だけ，ループされる．たとえば，次の場合には，ループは4回行われることになる．

```
array a = (1,2,3,4,5,6)
array b = (1,2,3,4)
scan a,b
  print a[]*b[]
/scan
```

● 関数

Scriptolでは，関数は関数名を定義する文で始まり，return文で終わる．たとえば，以下の文は，数値を受け取り，1を加算して表示するfuncという関数の定義である．

```
void func(number x)
    x + 1
    print x
return
```

関数名を定義する際には，C/C++と同様に戻り値の型，関数名，パラメータを指定する．ただし，Scriptolでは，戻り値として複数の値を指定することができる．たとえば，以下の関数は，戻り値として文字列一つと二つの数値を返す．

```
text, number, number testfunc
```

```
                    (number src1, number src2)
    src1 - 1
    src2 + 1
  return "TEXT", src1+src2, src1-src2
```

この場合，この関数を呼び出す場合にも，以下のように複数の変数を指定する．

```
text t
number a,b
t, a, b = testfunc( 10,5)
```

● クラス

Scriptolはオブジェクト指向の概念を取り入れており，クラスを定義することも可能である．クラスの定義は，class～/classというブロックで行う．

コンストラクタは，クラスと同じ名前のメソッドとして定義される．デストラクタはないようだ．これは，PHPにデストラクタが存在せず，変換できないためだろう．

```
class animal
  int age
  int getAge() return age
  void animal()
    age = 0
  return
  void addAge()
    age + 1
  return
/class
```

また，「is」というオペレータを利用することで，クラスの継承を行うこともできる．

```
class cat is animal
  boolean isWild
  void dog()
    isWild = true
  return
/class
```

さて，実際のクラスの使い方だが，オブジェクトはクラス変数を定義した際に生成される．

```
animal beast
print beast.getAge()
```

ただし，コンストラクタがパラメータをとる場合には，次のようにクラス名とパラメータを使って生成する．

```
dog dog1 = dog("pochi")
```

スクリプトのコンパイルと実行

作成したスクリプトは，C++かPHPに変換を行い，実行することができる．それぞれの変換は，「solc(C++への変換)」および「solp(PHPへの変換)」というプログラムで行う．

変換の方法は，コマンドラインからそれぞれのプログラムをパラメータに変換したいプログラムファイルを指定して呼び出すだけである．

```
solc test.sol
solp test.sol
```

すると，solcの場合は「test.cpp」(プログラムファイル)「test.hpp」(ヘッダファイル)の二つのファイルが，solpの場合は「test.php」というファイルが出力される．

また，solcでは，C++への変換の後，C++のコンパイラを起動して，コンパイルおよびビルドを行う機能も備えている．

次のように入力すると，ScriptolからC++への変換，C++のプログラムのコンパイルとビルド，実行までをまとめて行ってくれる．

```
solc -cbr test.sol
```

Scriptolが対応しているC++のコンパイラは，Borland社のBorland C++ Compilerと，MinGW[注2]に含まれるgccである．どちらが利用されるかは，「solc.ini」という初期設定ファイルで指定される．Scriptolがインストールされたディレクトリに置かれたsolc.iniは，Borland C++を利用する設定になっている．今回，筆者が検証した環境は，Borland C++があらかじめインストールされていたので，サンプルスクリプトのビルドは，まったく問題なくできた．

しかし，どうやらこの初期設定ファイルは，カレントディレクトリにおかれているものを読みにいくようなので，カレントディレクトリ(たいていは，ソースコードが置かれたディレクトリを利用するだろう)に，「solc.ini」を置いておかないと，その設定が反映されない．

solc.iniがない場合，solcはMinGWのgccを使う設定が標準になってしまうので，Borland C++しかインストールされていない環境では，エラーとなってしまう．サンプルスクリプトは，solcや標準のsolc.iniと同じディレクトリに置かれているので問題なくコンパイルできるが，自分で作成したスクリプトは独自のディレクトリで管理したい．

そこで，solc.iniをコピーし，中に書かれているインクルードファイルやライブラリの設定を書き換え，どこのディレクトリからでもきちんとビルドが行えるものを作成した(**リスト2**)．このファイルをソースファイルと同じディレクトリに置いておけば，どのディレクトリであっても，問題なくビルド作業を行うことができる．

さて，続いてPHPである．PHPの場合は，インタプリタがインストールされていれば，やはり変換したプログラムをそのまま実行することが可能である．その場合は次のように指定する．

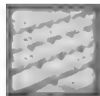
```
solp -r test.sol
```

注2：Minimalist GNU For Windowsの略．GCCでWindowsのプログラムを作成できるようにしたヘッダファイルとライブラリ群．http://www.mingw.org/

〔リスト2〕修正を行ったsolc.iniファイル

```
compiler=bcc32.exe
options= -a8 -c -C -4 -DBCC  -k -O1 -RT- -vi- -Vms -w-par -w-aus -w-use

-w-sig -w-ccc -Tkh30000 -tWC -q -I"c:¥program files¥scriptol"
linker=ilink32.exe
loptions= -C -Gn -x -c -q
objects=c0x32.obj "c:¥program files¥scriptol¥libsol.lib" "c:¥program

files¥scriptol¥gc.lib" import32.lib cw32.lib
extension = ".obj"
```

ちなみに，クライアントとして利用しているWindowsマシンの場合，PHPをインストールしていることはあまりないと思われるが，PHPのWindows版のバイナリファイルをダウンロードし[注3]，適当な場所にコピーを行い，php.exeにパスを通すだけで利用が可能になる．変換したスクリプトを直接PHPで実行するには，以下のように指定する．

```
php -q test.php
```

変換の実際

さて，Scriptolは，作成したソースコードはC++およびPHPに変換される．ScriptolとC++，PHPの言語仕様は，見た目の違いはあるものの，記述できることにはそれほどの違いがあるわけではないので，基本的には素直な変換が行われるようだ．

ただしScriptolは，変換先の言語に存在しない機能ももっている．そのため，それらの機能がどのように変換されるのかを調べてみた．

● 変数

まずは，存在しない変数型の問題である．まず，C++に存在しない変数型であるnumberは，doubleとして定義されている．そのほか，text，dyn，array，dictなどの変数型は，クラスとしてあらかじめ定義されたものがScriptolディレクトリに用意されており，それが利用される．それらのクラスの構造はそれぞれ，array.hpp，text.hppといったヘッダファイルで定義されている．そして，その実体はlibsol.a(もしくはlibsol.lib)というファイルに含まれており，実行ファイル作成時にまとめてビルドされる．

PHPの場合には，Scriptolのもつ変数の機能はほぼ網羅されており，とくにライブラリが呼ばれるようなことはない．

● 関数の戻り値

言語仕様の項でも述べたとおり，Scriptolは関数が複数の戻り値をとることができる．しかし，C++もPHPも，関数が返せる戻り値は一つである．そこで，複数の戻り値をもつ関数が，どのように変換されるのかを調べてみた．

テストに用いたサンプル，およびC++とPHPへの変換結果を**リスト3**に示す．結果，複数の戻り値があった場合は，arrayクラスにデータが格納され(変数名はdynクラスを使って格納されている)，戻り値としてそのarrayクラスが戻されている．arrayクラスやdynクラスが，実際の配列，連想配列だけでなく，内部的にも活用されていることがわかる．

〔リスト3〕テストに用いたサンプル，およびC++とPHPへの変換

```
text, int, int AddSub( int a, int b)
  int c = a + b
  int d = a - b
return "AddSub", c, d
```

(a) 複数の戻り値をもつ関数

```
array AddSub(int a,int b){
    int c=a+b;
    int d=a-b;
    return array(dyn("AddSub"))+dyn(c)+dyn(d);
}
```

(b) C++への変換結果

```
function AddSub($a,$b)
{
$c=$a+$b;
$d=$a-$b;
return array("AddSub",$c,$d);
}
```

(c) PHPへの変換結果

日本語の使用

Scriptolはフランスで作られている言語であり，これまで紹介してきた外国生まれのほとんどの言語と同様，日本語の利用はとくに考えられていないが，それは日本語が「使えない」という意味ではない．

たとえば，以下のような文字列として日本語を埋め込んだスクリプトを作成しても，問題なくコンパイル/実行することができる．とくに文字化けなどは起こらない．

```
text a = "あいうえお"
text b = "かきくけこ"
print a+b
```

ただし，Scriptolでは，「¥(アスキーコードで5Ch)」をエスケープ文字として利用している．そのため，文字コードとしてシフトJISを使っている場合には，注意が必要である．文字コードの一部として5Chが存在するからである．たとえば「噂(895Ch)」，「欺(8B5Ch)」といった文字などがある場合，文字の後にもう一つ「¥」をつけないと正しく表示できない．さらに，次の例のように，そのような文字が文字列の最後に来てしまった場合，C++/PHPへのコンパイル時にエラーが発生してしまう．最後のダブルクォーテーションが，直前の「¥(5Ch)」によって，エスケープされていると解釈されてしまうためである．

誤：`text title = "俺に関する噂"`

正：`text title = "俺に関する噂¥"`

注3：http://www.php.net/からダウンロードできる．

しかし，このエスケープ文字の問題は，C/C++でも同様に発生する問題であり，比較的よく知られているものなので，それほど大きな問題ではないだろう．したがって，Scriptolは，一般的な外国生まれのプログラミング言語程度には，日本語が利用できるということになる．

また，Windows版に付属するIDEでは，まったく日本語が使えない．初期状態では欧米文のフォントが指定されており，日本語が表示できない．IDEのフォントの設定は，同じディレクトリに置かれた「global.properties」というファイルの中で指定されている(**リスト4**)．しかし，そこで日本語が利用できるフォントを指定しても，やはり日本語を利用することはできなかった．どうやら，画面に表示を行う際に，1バイトずつ処理されてしまい，正しく表示されていないようである．残念ながら，IDEでの日本語の利用は，現時点ではあきらめざるを得ないようだ．

〔リスト4〕IDEの利用するフォントはglobal.propertiesで設定されている(抜粋)

```
# Give symbolic names to the set of fonts used
in the standard styles.
if PLAT_WIN
    font.base=font:Verdana,size:10
    font.small=font:Verdana,size:8
    font.comment=font:Comic Sans MS,size:9
    font.code.comment.box=$(font.comment)
    font.code.comment.line=$(font.comment)
    font.code.comment.doc=$(font.comment)
    font.text=font:Times New Roman,size:11
    font.text.comment=font:Verdana,size:9
    font.embedded.base=font:Verdana,size:9
    font.embedded.comment=font:Comic Sans MS,size:8
    font.monospace=font:Courier New,size:10
    font.vbs=font:Lucida Sans Unicode,size:10
if PLAT_GTK
    font.base=font:lucidatypewriter,size:12
    font.small=font:lucidatypewriter,size:10
    font.comment=font:new century schoolbook,size:12
    font.code.comment.box=$(font.comment)
    font.code.comment.line=$(font.comment)
    font.code.comment.doc=$(font.comment)
    font.text=font:times,size:14
    font.text.comment=font:lucidatypewriter,size:10
    font.embedded.base=font:lucidatypewriter,size:12
    font.embedded.comment=font:lucidatypewriter,size:12
    font.monospace=font:courier,size:12
    font.vbs=font:new century schoolbook,size:12
font.js=$(font.comment)
```

まとめ――Scriptolの存在意義

繰り返しになるが，Scriptolは，実際に実行可能なバイナリファイルを作成したり，インタプリタとしてスクリプトを実行することはできない．できることは，ほかの言語への変換だけである．

それなら，最初からC++やPHPで書けばよいのではないか，という意見が出てくることも当然予想される．もちろん，C++やPHPにすでに慣れ親しんでいて，それらの言語でプログラムを作成することがまったく苦でなければ，Scriptolを使う意義はあまりない．

だからといって，ScriptolをC++やPHPをまったく知らなくても，それらの言語でコードが書けるツール，ととらえることにもやや無理がある．というのも，作成したプログラムに問題があり，コンパイルや実行時にエラーが発生したような場合，まずは変換後のコードから問題を発見しなければならないようなことも考えられるからだ．そんなとき，もし変換後の言語についての知識がまったくなかったら，問題解決は相当難しくなってしまうだろう．

ではScriptolは，どのような人向けの言語なのだろうか．

Scriptolを使う最大のメリットは，C++に比べると，ずっと手軽に書くことができるという点である．言語仕様の策定において作者が決めた「七つのルール」にも，その思想ははっきりと現れている．したがって，Scriptolを使うのに向いているのは，C++も読むことはできるが，どちらかというと手軽にプログラミングを行いたい人なのだろう．また，PHPやPerlなどのスクリプト言語の経験があり，これからC++の書き方を勉強したいという人の勉強用のツールとしても役立つかもしれない．

この連載では，以前にBCX[注4]という，BASICをC言語に変換するツールを紹介している．こちらもCがわからない人向けというよりは，CもわかるけれどBASICでプログラミングがしたい人向けといったおもむきのツールだった．Scriptolも，やはりそれに近い位置付けができるのではないか，と筆者は感じた．

また，C++とPHP，両方に対応していることで，同じプログラムを両方の言語に変換することができるため，一度書いたコードの再利用性を高めるために利用するといった使い方もできるかもしれない．

最後にもう一つ，Scriptolを使ってみて感じたことがある．それは，Scriptolにはおそらく作者であるD.G. Sureau氏のコーディングにおける思想が，非常によく反映されているという点だ．プログラマはみなそれぞれ，コーディングに関する思想や美学をもっている．しかし，使用する言語によっては，その思想を完全にコードに反映することができないかもしれない．

しかし，新しい言語を作成してしまえば，自分の思想をそこに盛り込むのは，既存の言語を利用した場合よりも簡単だろう．しかも，コンパイラやインタプリタを作らず，コンバータという形にすることで，自分の求める言語を比較的容易に作成することができるのかもしれないということを，Scriptolが示しているような気がするのである．

みずの・たかあき

注4：http://bcx.basicguru.com/．詳細は，本誌2003年3月号の連載「開発環境探訪」を参照のこと．

Webサーバ機能をもつEthernet-シリアルコンバータ「XPort」活用技法

Javaアプレットによるシリアル機器の制御

川口幸裕

はじめに

前回(2003年6月号)では，XPort評価キットを使って，PC同士，PCとマイクロコンバータ，PCと加速度センサをそれぞれEthernetを経由してつなげてみた．実際，手元にあるシリアル機器が何の苦もなくつながったはずである(うまくつながらなかった方は，稿末のメールアドレスまでご一報いただきたい)．今回は最初に，前回の「おさらい」を含めてXPort評価キットの説明を簡単に行う．現在発売されているXPort評価キットは正式な製品であり，前回の紹介時のプロトタイプと大きな違いはない．XPort自体に変更はかかっておらず，評価ボード(**写真1**)も変更されていない．細かな違いを**表1**にまとめる．

本稿は，基本的には**表1**にも記した`XPort Sample Code and Solutions 910-816.pdf`の内容に沿って解説する．XPortに搭載されているWebサーバを使い，Javaアプレットを使用したシリアル機器との通信手段の詳細を述べる．マニュアルとサンプルコードの内容，実際に動作させるための手順・動作結果などを説明する．

Webページの設定について

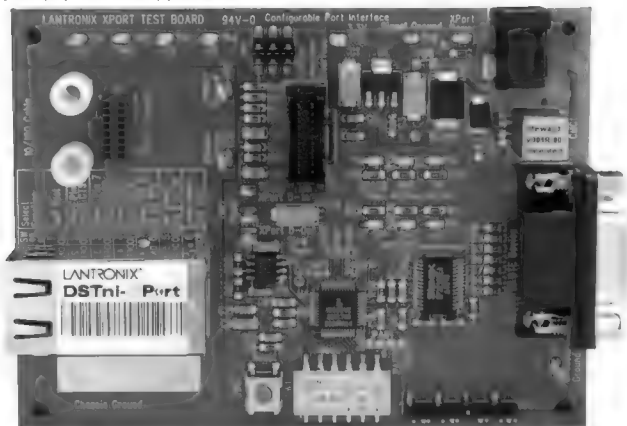
Javaアプレットの説明に入る前に，実際に作成したWebページをXPortにダウンロードする方法を説明する．ラントロニクスの製品でWebサーバを搭載したものはいくつかあるが，これらにおけるWebページデータの取り扱いについては，`.cob`ファイルというコンテナファイル形式を採用している．`.cob`形式は同社独自のものであり，`.htm`，`.html`を含むWebページの構成要素をすべて`.cob`ファイルに変換することで，取り扱いを簡単にしている．この`.cob`ファイルを作成する手順と，XPortへのダウンロード方法を説明する．なお`.htm`，`.html`ファイルの作成方法に関しては，それらに関する文献をご参照いただきたい．

● XPort InstallerによるWebページの設定

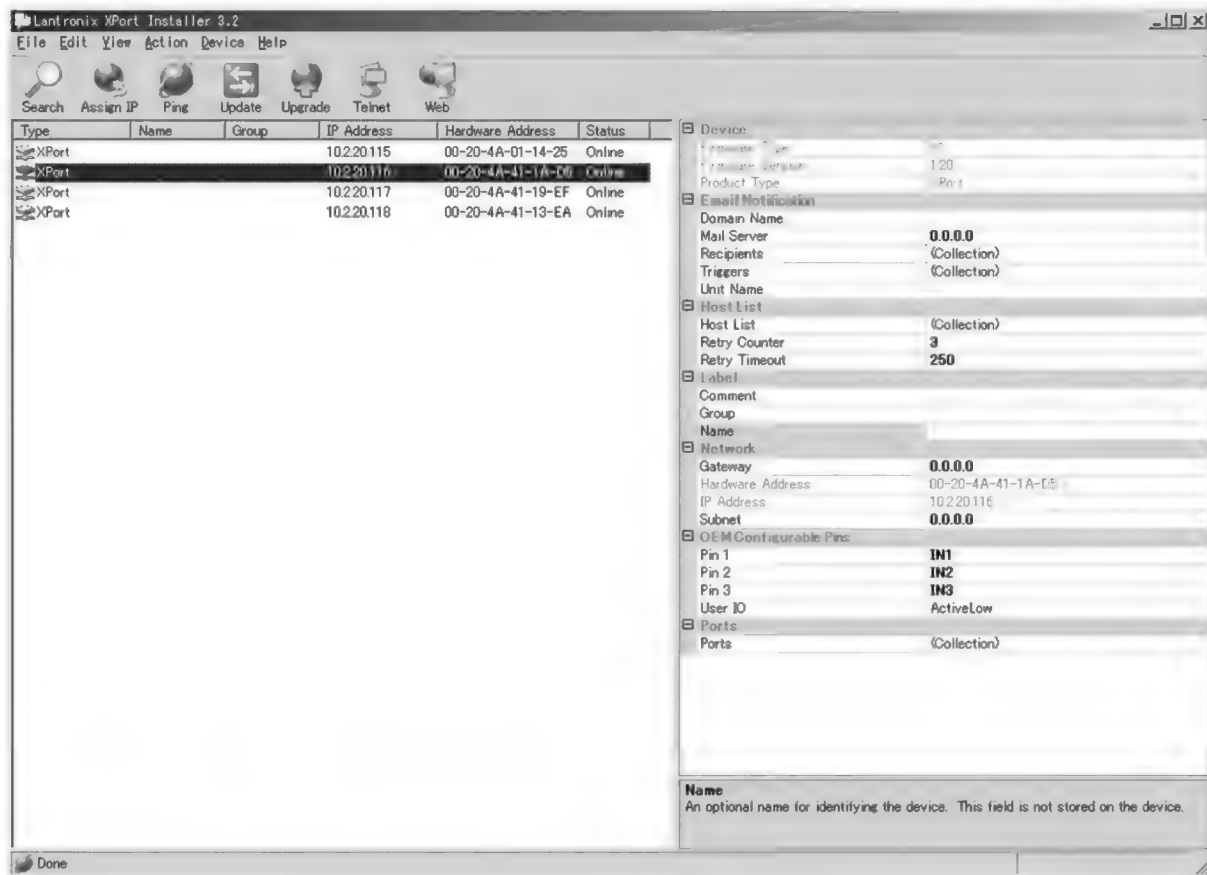
XPort Installerに内蔵されている`.cob`ファイル作成ツールで，実際に変換対象ファイル群が入ったディレクトリを指定するだけである(**図2**)．この場合，64Kバイト以内の大きさに収まる`.cob`ファイル1個の処理となり，`.cob`ファイルは自動的にXPortのWEB1エリアにダウンロードされる．

● WEB2COBによる.cobファイルの作成

XPortのフラッシュメモリは，64Kバイトごとのページ形式になっている．`.cob`ファイルが64Kバイトを超えるようであれば，フォルダを分け，別の`.cob`ファイルに変換する必要がある．また複数の`.cob`ファイルを作る場合，前述したXPort Installerではできない．そのため，WEB2COBという変換ユー

〔写真1〕XPort評価ボード

〔表1〕XPort評価キット——正式版とプロトタイプの違い

●添付されている「Device Installer」が「XPort Installer」に名称変更されている．また，Webページ，ファームウェアのダウンロード用のボタンが増えており(**図1**参照)，若干見た目が変わっている(機能そのものはまったく変わっていない)
●XPort内部のファームウェアが1.0 Beta 7から1.20にバージョンアップ
●マニュアル類が最新のものとなった(`Comm_Port_Redirector.pdf`，`XPort FAQs.pdf`，`XPort_DS_910-815.pdf`，`XPort_product_brief.pdf`，`XPort_UM_900-270.pdf`)——これらはラントロニクスのWebページ(`http://www.lantronix.com/`)から最新版をダウンロード可能
●`XPort Sample Code and Solutions 910-816.pdf`：Javaアプレットを使用したシリアル機器のコントロールに関してのマニュアル．このマニュアルはラントロニクス社のWebページには掲載されておらず，XPort評価キットに付属している．同時にこのマニュアルに沿ったサンプルアプリケーションが添付されている

〔図1〕XPort Installer

ティリティが用意されている．このプログラムはXPort Installerをインストールしたディレクトリに入っている．また，このプログラムはmimetype.iniというファイルを常に参照するので，かりに他のディレクトリで.cobファイルを作成したい場合，このファイルも一緒にコピーする必要がある．

サンプルコードとして用意されているものを別ディレクトリにコピーして試してみる．

C:¥TESTWEB　　　　　(**図3**)
C:¥TESTWEB¥WEBA　　(**図4**)
C:¥TESTWEB¥WEBB　　(**図5**)

WEB2COB.exeとmimetype.iniはC:¥TESTWEBにコピーしておく．データをWEBAとWEBBに分けているのは，64Kバイトに収まらないからである．またthermostat.zipは，実際に動作するJava Appletのコンテナファイルである．

WEB2COBを実行すると(**図6**)，WebA.cob，WebB.cobという二つのファイルができあがる．

● Webページのダウンロード

できあがった.cobファイルは，XPort InstallerでXPortにダウンロードする(**図7**)．

また，TFTPクライアントによるWebページのダウンロードも可能である．Windows 2000/XPにはTFTPクライアント

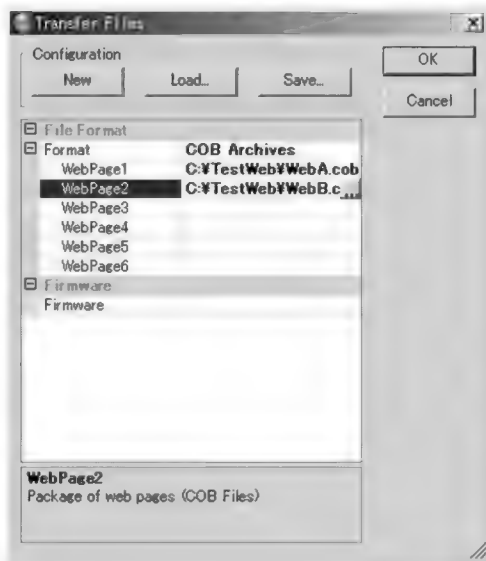
〔図2〕XPort Installerによる.cobファイルへの変換

〔図3〕TESTWEBディレクトリの内容

〔図4〕WEBAディレクトリの内容

〔図5〕WEBBディレクトリの内容

〔図6〕WEB2COBの実行

〔図7〕.cobファイルをXPort InstallerでXPortにダウンロード

が装備されている．これを使い，XPort Installerを使わないで.cobをインストールすることもできる．

```
C:¥TESTWEB> tftp -i 10.2.20.115 PUT WebA
                                   .cob WEB1
C:¥TESTWEB> tftp -i 10.2.20.115 PUT WebB
                                   .cob WEB2
```

XPort側のファイル名として，WEB1，WEB2～WEB6までを指定する．これは，それぞれXPort上のフラッシュメモリのページに相当する．

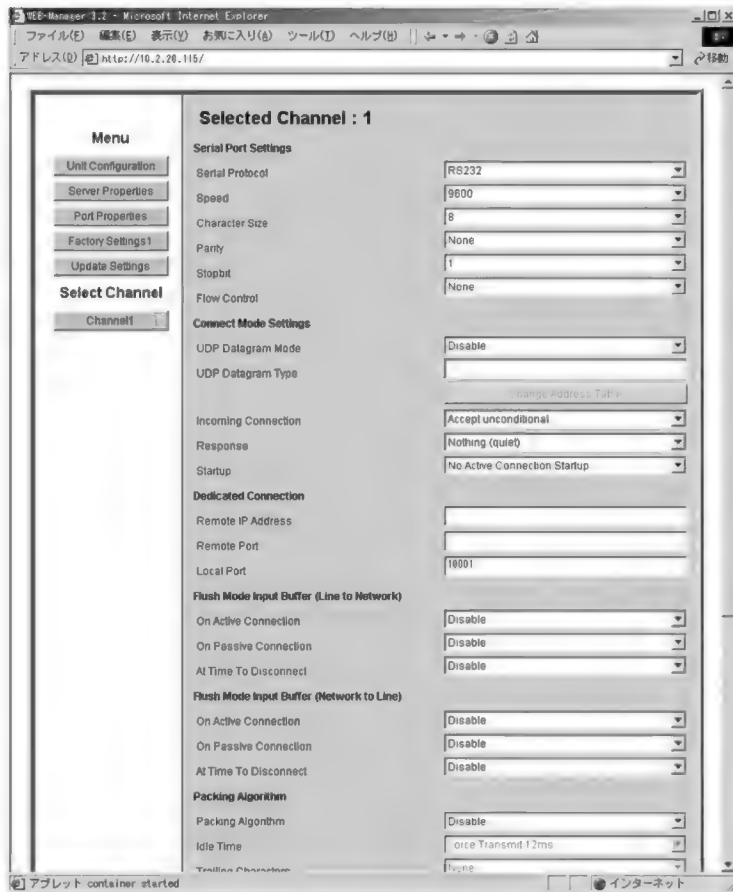
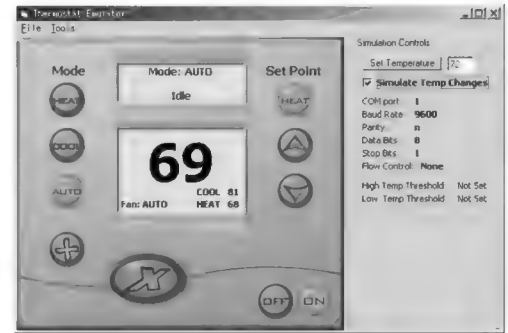
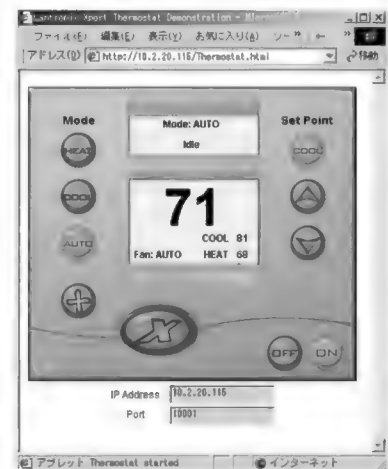
ここまでで，WebページをXPortにダウンロードする方法は理解いただけたと思う．難しい作業ではないので，ツールの使い方さえ間違えなければ，問題なくWebページの設定ができるはずだ．

● Web Manager

XPortには標準で，Web ManagerがデフォルトのWebページとして登録されている（**図8**）．このWebページはブラウザを通して，XPortの諸情報を変更することができる．実際にこれ自体もJavaアプレットで記述されている．

Java Applet

それでは本題のJava Appletを使ったシリアル機器のコントロールを行ってみよう．サンプルとして用意されているのは，先ほど使用したWebA.cob，WebB.cob，それとPC側のアプ

〔図8〕**Web Manager**

〔図9〕**RS-232-Cポートをもった PCでアプリケーションを動作させる**

〔図10〕**Java Appletの起動**

リケーションである．実際にこのサンプルアプリケーションを動作させるために，PCを2台用意する．1台はRS-232-Cポートが必須である．もう1台はネットワークインターフェースをもったもので，ブラウザが動作する環境が必要である（RS-232-Cポートとネットワークインターフェースを両方もったPC1台でもかまわない）．

RS-232-Cポートをもった PC側のアプリケーションはVisual Basicで記述された室温制御をシミュレーションするプログラム（ファイル名：thermostat.exe）で，RS-232-Cから送られたデータを反映させ，また変更がかかったときに，そのデータをRS-232-Cに送出するものである．このアプリケーションで制御するものは，コントローラのオン/オフ，暖房/冷房/オートモードの設定，ファンのオン/オフ，冷房/暖房のスイッチを入れるための温度（しきい値）の設定となっている．このアプリケーションでPCは仮想のシリアル室温コントローラとなる．

2台のPC間をXPort評価ボードで接続する．XPort評価ボードのRS-232-Cポートを同じくRS-232-Cポートをもった PCにストレートケーブルで接続する．XPort評価ボードのEthernetをもう1台のネットワークインターフェースをもった PCに接続する．直結するのであればクロスのEthernetケーブル，ハブを介するのならストレートEthernetケーブルでかまわない．

RS-232-Cポートをもった PCで，先ほどのVisual Basicで記述されたアプリケーションを動作させる（**図9**）．ボタンを押すと温度設定や，ファンのスイッチが切り替えられる．この状態で，Ethernet側のPCでXPort Installerを起動し，ポートには10001を指定してTelnetを立ち上げる．するとコンソール画面に3文字のデータが表示される．これは室温コントローラから送られてくる現在のステータスである．実際に室温コントローラを操作すると，この3文字のステータスデータが変化する．この段階では，前回述べたとおりシリアル機器の情報をEthernet接続されたTelnet端末で受信できていることを意味する．

続いて，ネットワーク接続されているPC側でブラウザを起動する．アドレスにxxx.xxx.xxx.xxx/thermostat.htmlと入力する（xxx.xxx.xxx.xxxはXPortに割り付けたIPアドレス）．室温制御アプリケーションとほぼ同じ画面のJava Appletが起動する（**図10**）．

室温制御アプリケーション，ブラウザ側のJava Appletのどちらのステータスを変更しても互いに同期が取られ，同じステ

その他のJava Applet

今回の新しい評価ボード用CD-ROMに含まれているサンプルアプリケーションには，XPortを使うためのJava Appletが収録されている．XPort用ユーティリティとしてCBXConfig.Javaというファイルが含まれており，その中には**図A**のようなものが含まれている．これらはXPortをさらに使いこなすために用意されているもので，これがなければXPortの基本機能を簡単に使いこなせないというわけではない．

〔図A〕CBXConfig.Javaの内容

・XPort設定ユーティリティ

`public void setEmailConfig()`	メール発信機能の設定
`public void reboot()`	リブート
`public CBXConfig(InetAddress ipa, int port)`	IPアドレスとポートの設定
`public CBXConfig(InetAddress ipa)`	IPアドレスの設定
`public void disconnect()`	ディスコネクト
`public int monitor(int portn)`	モニタリング
`public byte setRemoteIP(String rip, int port, int mode)`	リモートIPの設定
`public void reset()`	リセット
`public void setConfig()`	コンフィグレーションの反映

・その他文字列操作

```
public int stringToPort(String _given), private byte[] stringToIp(String _given),
private int toInt(byte given)
private String byteToHex(byte b), private String charToHex(char c)
```

・サンプルコード

```
public void reboot()
{
   ctp.disconnect();
   sleeper(5);
}

public void reset()
{
   int i;

   data[0] = 0x00;
   data[1] = 0x00;
   data[2] = 0x00;
   data[3] = (byte) RESET;

   byte[] btmp = Password.getBytes();

   data[4] = btmp[0];
   data[5] = btmp[1];

   ctp.send(data);
   ctp.disconnect();
   sleeper(5);
}

public void setConfig()
{
   int i;

   odata[0] = 0x00;
   odata[1] = 0x00;
   odata[2] = 0x00;
   odata[3] = (byte) PUTSU;

   for(i = 0; i < 120; i++)
   {
      odata[i+4] = nSetup[i];
   }

   ctp.send(odata);
   ctp.disconnect();
   sleeper(5);
}

public int stringToPort(String _given)
{
   int temp = sti(_given);
   if((temp > 0) && (temp <= 0xffff))
   {
      return (temp);
   }

   return(-1);
}

private byte[] stringToIp(String _given)
{
   byte[] retval = new byte[5];
   retval[4] = -1; // setting error
   StringTokenizer st = new
      StringTokenizer(_given, ".");

   if (st.countTokens() != 4)
      return(retval);

   int[] temp = new int[4];

   for (int i = 0; i < 4; i++)
   {
      temp[i] = sti(st.nextToken());
      retval[i] = (byte) (temp[i] & 0xff);

      if (temp[i] != toInt(retval[i]))
         return(retval);
   }

   retval[4] = 0;
   return(retval);
}

public void sleeper(int given)
{
   try
   {
      Thread.sleep(given*1000);
   }
   catch (Exception ex)
   {
      System.out.println("ERROR in sleeper("
         + given + "):" + ex);
   }
}
```

ータスになる．これにより，Java AppletとRS-232-Cを介してアプリケーションが通信を行っていることが確認できる．なお，以降掲載するリストはXPort評価キットとサンプルに付属しているCD-ROMからの抜粋であり，コメントは筆者が日本語化している．

実際にXPortを介してシリアル側にデータを送るためのJava Appletソースコードは，**リスト1**のとおりである．

この`tcpip.java`は全体では200行ほどで，Java AppletからのTCP/IPのコネクション，データの送受信などが含まれている．XPortを通したシリアル機器への通信のための機能はすべて網羅されている．アプリケーションからのコーリングシーケンスは**リスト2**のようになる．

この二つのアプリケーション，Visual Basic版とJava Applet版は，3文字のデータを送受信することで同期を取っている．実際に暖房/冷房などのスイッチに関しては，3文字のデータの先頭バイトによりケース分けをしており，実際の温度に関してはその後の2文字のデータで通信を行っている．このように，実アプリケーション(シリアル機器)との通信を確立することで，Java Appletのようなブラウザレベルでのコントロールが可能となる．これはXPortに内蔵されたWebサーバを使ったアプリケーション事例の一つととらえていただきたい．

Java Appletを使ったシリアル機器の制御は意外と簡単だということを理解していただけたのではないだろうか．サンプルコードもふんだんに用意されており，Javaに精通した方ならすぐにアプリケーション開発ができると思う．また，XPortへの設定ユーティリティも用意されており(**コラム1**参照)，これらを使用することで，より複雑なアプリケーションも短期間で開

〔リスト1〕XPortを介してシリアル側にデータを送るためのJava Appletソースコード(`tcpip.java`より抜粋)

```
import java.net.*;
import java.util.*;
import java.io.*;
import java.*;

public class tcpip
{
   protected Socket s = null;

   public DataInputStream dis = null;
   protected DataOutputStream dos = null;

   public synchronized void send(byte[] temp)
   {
      try
      {
         dos.write(temp, 0, temp.length);
         dos.flush();
      }
      catch(Exception ex)
      {
         System.out.println("Error sending data : "
            + ex.toString());
      }
   }
 .
 .
}
```

コラム2

ラントロニクス社のその他製品

ラントロニクス社は，XPort以外にも「箱もの」と呼ばれるデバイスサーバ，ターミナル/コンソールサーバを用意している．デバイスサーバに関しては，基本的な機能はXPortと同等である．RS-232-C/422/485インターフェースをもち，これらの対応機器をEthernetに接続できる．XPortはシリアル機器への内蔵を前提に開発されているが，場合によっては外付けのほうが好ましい場合にはこのようなデバイスサーバをおすすめする(**写真A**)．

ターミナル/コンソールサーバは，シリアル1対ネットワーク1の構成であるデバイスサーバに比べ，複数のシリアルポートをもったものである．4ポートから最大48ポートまでもっており，アプリケーションにあわせた選択肢が広くなっている．これらは一つのEthernetポートをもち，複数のシリアルポートを一括でコントロールできる．具体的には複数のXPortとEthernetハブが一つの筐体に入ったものだが，複数ポートを一括コントロールするためのより高いパフォーマンスをもち，セキュリティやサポートされているネットワークプロトコルはデバイスサーバに比べて広範囲に対応する(**写真B**)．

〔写真A〕外付けデバイスサーバの例(UDS100)

〔写真B〕ターミナル/コンソールサーバの例(ETS16P)

〔リスト2〕アプリケーションからのコーリングシーケンス（thermostat.javaより抜粋）

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.applet.Applet;
import java.io.*;
import java.net.*;
import java.util.*;
import netscape.javascript.*;

class ThermostatCanvas extends Canvas implements
MouseListener
{
.
.

   // IP stuff
   public tcpip gtp;

.
.

   public synchronized void sendHeatingSetpoint()
   {
      // このファンクションは暖房のスイッチを入れるため
      // の温度設定をXPortを通して室温コントロール
      // アプリケーションに送るものです．

      if (gtp != null)
      {
         String s = new String("H" +
         new String(new Integer(iHeatingSetpoint).toString())
         + "¥r");
         gtp.send(s);
      }
   }
.
.
}
```

発できる．

おわりに

前回記したSDKに関する解説は，残念ながらリリースが原稿執筆に間に合わなかったため，解説することはかなわなかったが，本号が発売される頃にはリリースされているはずである．

XPortは，CPU/メモリ/入出力装置を装備した一つのシステム製品である．内蔵可能でかつプログラマブルということで，シリアル機器に対するプログラマブルコントローラとして使うことも十分考えられる．最近ではフラッシュメモリにデータを蓄えるデータロガー，シリアル機器同士を直接結ぶコントローラなど，さまざまな用途が考えられている．これらの機能を実現するためには，やはりSDKの存在は無視できない．Java Appletは入出力に関する選択肢を広げるものだが，機能そのものの拡張にはやはりSDKが必要だと思う．リリースされしだい，このあたりの詳細を調査する予定である．興味のある方は以下のメールアドレスへご連絡いただきたい．

■「XPort」に関する問い合わせ先
橘テクトロン（株）
●営業関係問い合わせ先：sales.lantronix@tachitec.co.jp
●技術的ご質問：support.lantronix@tachitec.co.jp

かわぐち・ゆきひろ　橘テクトロン（株）

音楽配信技術の最新動向

ピアツーピアを利用した配信とOggVorbisのエンコード

第5回

岸 哲夫

前回は，OggVorbisファイルをデコードするプログラムについて説明しました．今回からは，OggVorbisファイルを作るプログラム(エンコーダ)を作ってみましょう．

その前に，インターネット上で見つけた，ピアツーピアを利用した音楽配信についての話題を少し取り上げます．

ピアツーピアを利用した配信

本連載でも何度か取り上げましたが，インターネットを利用した音楽配信の先駆者であるミュージシャンの平沢進氏が，ネット上でイベントを行いました．

平沢進氏はコンピュータを介し，観客の意志が次々と反映され，ステージと観客が対話しながら進行する「インタラクティブライブ」を行ってきましたが，このイベントも，その宣伝とテストを兼ねて行ったようです．

いままでにも会場の観客だけではなく，ネット上から接続した「在宅オーディエンス」ともWebアプリを通じて対話する実験をしてきましたが，それに加えて今回のライブでは，より大がかりにピアツーピア方式でライブを中継する計画です．ライブ本番は5月の連休中に行われます．どのように進められたかは次号で報告します．

インタラクティブライブを説明しているWebサイトを**図1**に，今回のインタラクティブライブ「LIMBO-54」のWebサイトを**図2**に示します．

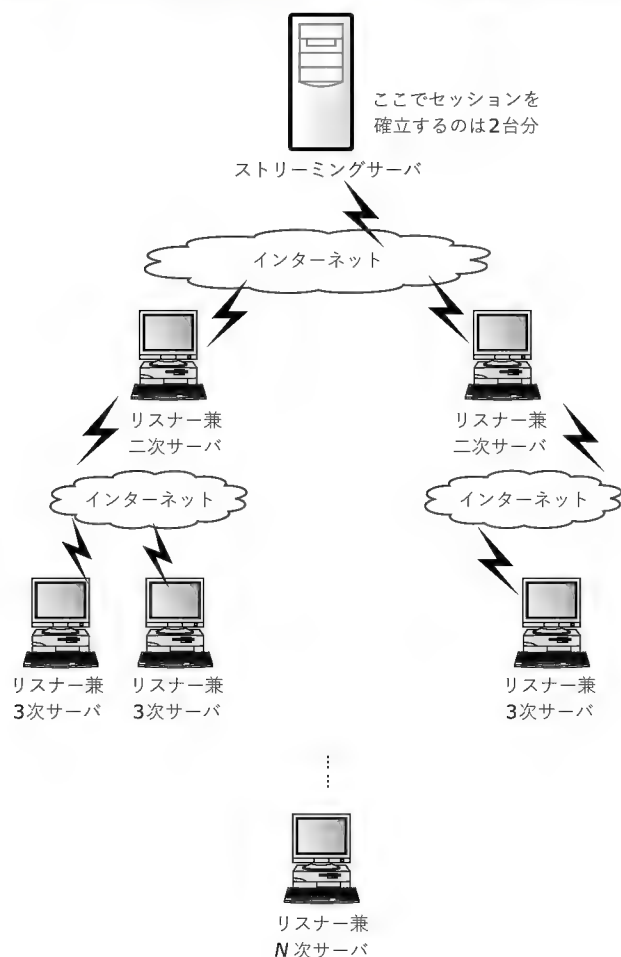
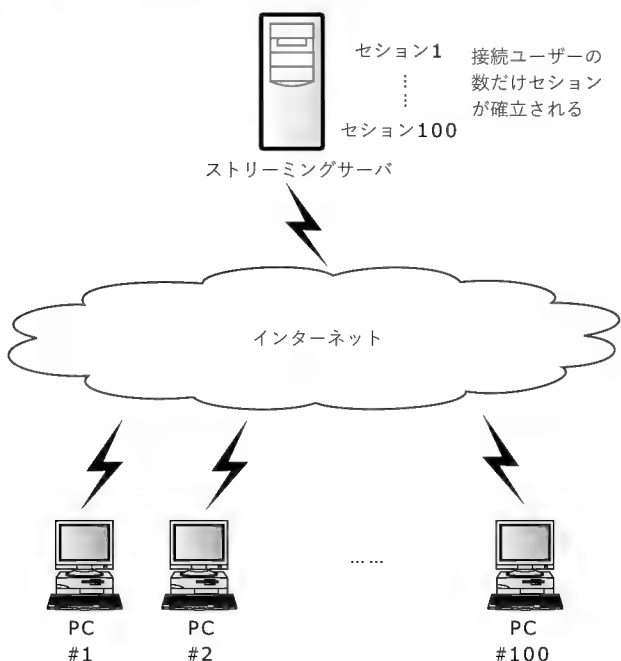
今回はストリーミングを中継するソフトとしてSHOUTcast，PeerCastを使用することを推奨しています．

通常，ストリーミングというとサーバがあり，そこに数多くのユーザーが接続し，サーバはユーザーの接続数だけセッションを確立しなければなりません．

ピアツーピアの場合，発信元のサーバは2次サーバにセッションを確立するだけで済みます．2次サーバは3次サーバにセッションを確立するだけです．かりに2ユーザーずつ発信元サーバ→その次→その次と理想的に接続すれば，N次サーバの時点で2のN乗だけのユーザーが音楽を聴くことができます．

〔図1〕What is the interactive live ?
(http://plaza20.mbn.or.jp/~ghost/intera.htm)

〔図2〕Susumu Hirasawa Interactive Live Show 2003「LIMBO-54」
(http://premium.nikkeibp.co.jp/il2003/index.shtml)

〔図3〕ピアツーピアを使用したストリーミングの概念

〔図4〕通常のストリーミングの概念

たとえば16次サーバまで接続できたとしたら，65535 −1人が音楽を楽しむことができます(**図3**)．通常の方式でそれだけの人数がアクセスすると，回線の太さやサーバの容量を考えなければなりません(**図4**)．ピアツーピアの場合，それが大きな利点となります．

ピアツーピア方式のストリーミング配信技術については，次号で詳細に説明する予定です．

OggVorbisのエンコード

さて，今回の本題であるOggVorbisのエンコードについて説明します．今回は，そのために必要となるlibvorbisenc APIを説明します．

libvorbisencの現行バージョンは非常に単純な構造です．エンコードエンジンを適切にセットアップする初期化機能を含み，今後の変更にも対応する，エンコーダセッティングのコントロール機能も含んでいます．

libvorbisencルーチンは，すべて「vorbis/vorbisenc.h」の中で宣言されます．

● 関数： vorbis_encode_init

この関数は，その名のとおり初期化処理をします．必要な構造体にパラメータを設定し，適切なエンコードができる環境を構築します．

なお，この関数は固定ビットレートを用いるときに使います．

ここで指定する最大ビットレートや最小ビットレートは一種のコメントとして，作成されたエンコードデータに記録されます．

使う側がビットレートの最大値や最小値として使用します．

この処理の前に，libvorbis APIの初期化処理であるvorbis_info_init()を呼んでおきます．そしてもちろん，エンコードの後には，vorbis_info_clearを呼ばなくてはなりません．

プロトタイプを**リスト1**に示します．

以下，引き数を説明します．

▶ vorbis_info *vi

初期化したいvorbis_info構造体を指定します．

▶ long channels

エンコードしたいチャネル数をセットします．

▶ long rate

入力音源のサンプリングレートを指定します．

〔リスト1〕vorbis_encode_initのプロトタイプ

```
extern int vorbis_encode_init(vorbis_info *vi,
                              long channels,
                              long rate,

                              long max_bitrate,
                              long nominal_bitrate,
                              long min_bitrate);
```

〔リスト2〕vorbis_encode_init_vbrのプロトタイプ

```
extern int vorbis_encode_init_vbr(vorbis_info *vi,
                                  long channels,
                                  long rate,
                                  float base_quality /* from 0. (lo) to 1. (hi) */);
```

〔リスト3〕vorbis_encode_ctlのプロトタイプ

```
extern int vorbis_encode_ctl(vorbis_info *vi,int number,void *arg);
```

▶ long max_bitrate

エンコードデータに記録される最大ビットレートです．これはこのデータを使う側が，ビットレートの最大値として使います．

▶ long nominal_bitrate

この値で指定されたビットレートでエンコードされます．

▶ long min_bitrate

エンコードデータに記録される最小ビットレートです．これは，このデータを使う側が，ビットレートの最小値として使います．

そして，戻り値がゼロ以外ならエラーです．

● 関数：vorbis_encode_init_vbr

この関数は，その名のとおり可変ビットレート用に初期化処理をします．必要な構造体にパラメータを設定し，適切なエンコードができる環境を構築します．

この処理の前には，やはりlibvorbis APIの初期化処理であるvorbis_info_init()を呼んでおきます．

もちろん，エンコードの後にはvorbis_info_clearを呼ばなくてはなりません．

プロトタイプを**リスト2**に示します．

以下，引き数を説明します．

▶ vorbis_info *vi

初期化したいvorbis_info構造体を指定します．

▶ long channels

エンコードしたいチャネル数をセットします．

▶ long rate

入力音源のサンプリングレートを指定します．

▶ float base_quality

0から1の間の正の値でエンコード品質を指定します．

そして，戻り値がゼロ以外ならエラーです．

● 関数：vorbis_encode_ctl

公式マニュアルにはまだ実装されていないと書いてありますが，すでにリリースされているようです．

この関数を使ってエンコードを行います．

プロトタイプを**リスト3**に示します．

以下，引き数を説明します．

▶ vorbis_info *vi

初期化済みでパラメータをセットしたvorbis_info構造体を指定します．

▶ int number

エンコードの際には定数OV_ECTL_RATEMANAGE_GETをセットするようです．値は0x10です．

▶ void *arg

構造体ovectl_ratemanage_argを指定します．ここに処理されたサンプリングレートなどが戻ります．実際のサンプルプログラムを作成する際に詳しく検証します．

そして，戻り値がゼロ以外ならエラーです．

＊　　＊

次回は，libvorbisenc APIを使って簡単なサンプルプログラムを作成してみます．

きし・てつお

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第⑲回　制御転送命令

大貫広幸

これまで，x86系CPUの汎用命令に分類される転送，演算，ビット/バイト/フラグに関する命令について説明してきました．x86系CPUの汎用命令も，残るはジャンプ/コールに関する命令と，ストリング命令と呼ばれるバイト列を扱う命令群のみです．そこで今回は，ジャンプ/コールに関する命令について説明することにします．

x86系CPUでは，ジャンプ/コールのようなプログラムの実行を制御する命令のことを「制御転送命令」と呼びます．

プログラムの実行を制御する命令の種類

x86系CPUのプログラムの実行を制御する命令としては，**表1**に示すような「制御転送命令」のほかに，「その他の命令」に分類される命令も含まれます．

● 制御転送命令の種類

制御転送命令は，プログラムの実行を制御するための命令で

〔表1〕x86系32ビットCPUのプログラム実行制御に関する命令

分類	インストラクション名	動作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
制御転送命令	JMP	Jump 指定されたアドレスに無条件でプログラムの制御を移す	·	·	·	·	·	·
	Jcc	Jump if Condition Is Met 条件ccが成立していれば指定されたアドレスにプログラムの制御を移す	·	·	·	·	·	·
	LOOP LOOPcc	Loop with (E)CX counter レジスタ(E)CXを－1し，ゼロでなければ指定されたアドレスにプログラムの制御を移す．条件ccが指定されていた場合は，上記の条件の他に指定条件ccも成立していれば，指定されたアドレスにプログラムの制御を移す	·	·	·	·	·	·
	CALL	Call Procedure リターンアドレスをスタックにPUSHし，指定されたアドレスに無条件でプログラムの制御を移す	·	·	·	·	·	·
	RET	Return from Procedure スタックからリターンアドレスをPOPし，そのリターンアドレスに無条件でプログラムの制御を移す	·	·	·	·	·	·
	INT	Software Interrupt ソフトウェア割り込みを発生させる	· IF－0，TF－0	·	·	·	·	·
	INTO	Software Interrupt on overflow OF=1なら4番のソフトウェア割り込みを発生させる	· IF－0，TF－0	·	·	·	·	·
	BOUND	Check Array Index Against Bounds 配列インデックスが上下限を超えていないかどうか調べ，範囲外なら5番のソフトウェア割り込みを発生させる	·	·	·	·	·	·
	IRET	Interrupt Return 割り込み処理ルーチンから通常のルーチンに戻る	* IF－*，TF－*	*	*	*	*	*
	ENTER	Make Stack Frame for Procedure Parameters 高級言語のためのスタックフレームを作成する	·	·	·	·	·	·
	LEAVE	High Level Procedure Exit ENTER命令で作成したスタックフレームを削除する	·	·	·	·	·	·
その他の命令	NOP	No Operation 何の操作もしない命令	·	·	·	·	·	·
	UD2	Undefined Instruction 無効オペランドの例外(割り込み番号6)を発生させる命令	·	·	·	·	·	·

注：表中の影響を受けるフラグの記号は次の状態を表す
・＝変化なし
＊＝結果にしたがって変化する
0＝クリアされる

す．この命令には，

(1) 指定アドレスに無条件で移行する「無条件ジャンプ」
(2) CPUが指定の状態にあるとき，指定したアドレスに移行する「条件付きジャンプ」
(3) 指定回数プログラムの同じ部分を繰り返し実行する「ループ」
(4) サブルーチンを呼び出すための「コール」とサブルーチンから元のルーチンに戻るための「リターン」
(5) ソフトウェア割り込みおよび例外を発生させる命令と割り込み例外からのリターン命令
(6) サブルーチンのためのスタックフレームの作成と解放

の6種類があります．

(1)～(5)は移行命令なので，CPUのレジスタEIPの書き換えを行います．しかし(6)の命令は，サブルーチンで使われるスタックフレームの操作なので，レジスタESPとEBPが操作対象になり，レジスタEIPの変更はありません．

● 「その他の命令」に分類されている命令

「その他の命令」の中で実行に関する命令としては，NOP命令とUD2命令の2命令があります．

NOP命令は，“No Operation”の略で，その名が示すように，NOP命令を実行してもCPUは何の操作も行いません．NOP命令自体はオペランドがなく，オペコードが90hの1バイト長の命令です．そのため，このNOP命令を実行してもレジスタEIPが＋1されるだけで，すべてのフラグ，レジスタ，メモリ，I/Oに影響を与えません．

UD2命令は，“Undefined Instruction”といい，無効オペコードの例外を発生させます．UD2命令にもオペランドはありません．このUD2命令は，無効オペコードの例外(割り込み番号6)を発生する以外は，NOP命令と同じでCPUに何の影響も与えません．UD2命令の用途は，無効オペコードの例外を処理するハンドラをデバッグするときに使用するもので，通常の多くのプログラムでは，このUD2命令を使いません．そのため，この連載で使用しているアセンブラMASMもgasも，このUD2命令はサポートしていません．

ジャンプ命令(JMP，Jcc)

ジャンプ命令のニモニックは，無条件ジャンプ命令の場合は「JMP」です．状態付きジャンプ命令の場合は，**表2**に示す条件を表す文字をccとすると，頭にJを付けて「Jcc」と記述します．

JMP命令は，無条件に指定された飛び先の命令にジャンプします．JMP命令には，飛び先のアドレスの違いから，同じ物理セグメント内なら「nearジャンプ」，異なる物理セグメントなら「farジャンプ」を使うことになります．また，nearジャンプはさらにアドレスの指定方法の違いから「相対オフセット」，「絶対オフセット」の2種類に分けられます．

Jcc命令は，ccで指定された条件(**表2**)が成立した場合のみ，指定された飛び先の命令にジャンプします．飛び先のアドレスは，同じ物理セグメント内のみで「相対オフセット」で指定します．

〔表2〕Jcc命令のニモニックで使われる条件を表す文字

文字	内　容	成立となる条件
CXZ	レジスタCXがゼロ (shortジャンプのみ使用可能)	CX=0
ECXZ	レジスタECXがゼロ (shortジャンプのみ使用可能)	ECX=0
E	Equal ……等しい	ZF=1
Z	Zero ……ゼロ	ZF=1
NE	Not Equal ……等しくない	ZF=0
NZ	Not Zero ……ゼロではない	ZF=0
A	Above ……より上	CF=0 and ZF=0
NBE	Not Below or Equal ……より下でなく等しくない	CF=0 and ZF=0
AE	Above or Equal ……より上か等しい	CF=0
NB	Not Below ……より下でない	CF=0
B	Below ……より下	CF=1
NAE	Not Above or Equal ……より上でなく等しくない	CF=1
BE	Below or Equal ……より下か等しい	CF=1 or ZF=1
NA	Not Above ……より上でない	CF=1 or ZF=1
G	Greater ……より大きい	ZF=0 and SF=OF
NLE	Not Less or Equal ……より小さくなく等しくない	ZF=0 and SF=OF
GE	Greater or Equal ……より大きいか等しい	SF=OF
NL	Not Less ……より小さくない	SF=OF
L	Less ……より小さい	SF ≠ OF
NGE	Not Greater or Equal ……より大きくなく等しくない	SF ≠ OF
LE	Less or Equal ……より小さいか等しい	ZF=1 or SF ≠ OF
NG	Not Greater ……より大きくない	ZF=1 or SF ≠ OF
C	Carry ……キャリーがある	CF=1
NC	Not Carry ……キャリーがない	CF=0
O	Overflow ……オーバフローがある	OF=1
NO	Not Overflow ……オーバフローがない	OF=0
S	Sign ……符号がある(負数)	SF=1
NS	Not Sign ……符号がない(非負数)	SF=0
P	Parity ……パリティがある	PF=1
PE	Parity Even ……パリティが偶数	PF=1
NP	Not Parity ……パリティがない	PF=0
PO	Parity Odd ……パリティが奇数	PF=0

● 相対オフセットによるnearジャンプ

相対オフセットとは，アドレス上，JMP/Jcc命令の次に配置された命令の先頭アドレスをゼロと考え，飛び先の命令までのバイト数を指定するものです(**図1**)．

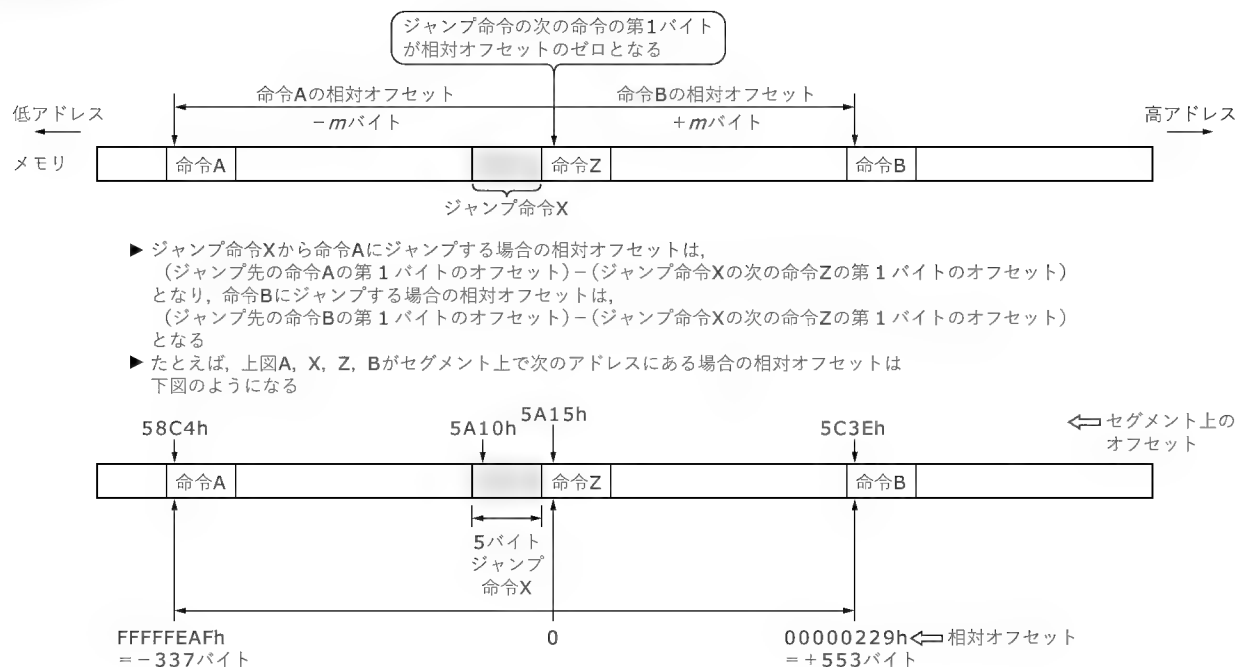
そのため，相対オフセットの値は符号付き整数となります．ジャンプのときは，

レジスタEIP←ジャンプ命令の次の命令のオフセット
＋相対オフセット

という計算を行ってジャンプします．このとき，プログラムが16ビットプログラムで実行していた場合，レジスタEIPには「ジャンプ命令の次の命令のオフセット＋相対オフセット」の下位

〔図1〕相対オフセット

▶ ジャンプ命令Xから命令Aにジャンプする場合の相対オフセットは，
（ジャンプ先の命令Aの第1バイトのオフセット）−（ジャンプ命令Xの次の命令Zの第1バイトのオフセット）
となり，命令Bにジャンプする場合の相対オフセットは，
（ジャンプ先の命令Bの第1バイトのオフセット）−（ジャンプ命令Xの次の命令Zの第1バイトのオフセット）
となる

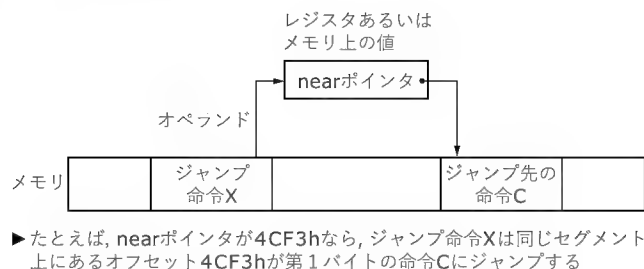
▶ たとえば，上図A，X，Z，Bがセグメント上で次のアドレスにある場合の相対オフセットは
下図のようになる

ワード（レジスタ IP）のみにジャンプ先オフセットがロードされ，上位ワードはゼロになります．

相対オフセットの値は，機械語命令の中にその値が埋め込まれます．相対オフセットによる near ジャンプでは，セグメント内全域をジャンプ対象とする「相対 near ジャンプ」と，ごく限られた範囲をジャンプ対象とする「相対 short ジャンプ」の2種類があります．

また，とくにごく限られた範囲をジャンプ対象とするジャンプ命令のことを単に「short ジャンプ」といい，区別しています．short ジャンプは，相対オフセットの値が1バイトの範囲（−128～+127）と決められています．そのため，short ジャンプはジャンプできる範囲は狭いものの，機械語命令の長さで比べた場合，もちろん short ジャンプのほうが，セグメント内全域をジャンプ対象とした相対オフセットの near ジャンプより短くなります．

Jcc 命令のうち，JCXZ，JECXZ 命令は，short ジャンプしかありません．そのため，JCXZ，JECXZ 命令のみ相対オフセットが−128～+127の範囲内でのジャンプとなります．

〔図2〕絶対オフセット

▶たとえば，nearポインタが4CF3hなら，ジャンプ命令Xは同じセグメント上にあるオフセット4CF3hが第1バイトの命令Cにジャンプする

アセンブラ MASM も gas も，飛び先である命令までのバイト数の違いにより，short ジャンプにするか否かを自動的に判断して，機械語命令を生成するため，プログラミングの段階では，この short ジャンプにするか否かの違いは，あまり考える必要はありません．ただし，今述べたように JCXZ，JECXZ 命令は short ジャンプしかないため，この JCXZ，JECXZ 命令のみ，飛び先の命令までのバイト数には，注意する必要があります．

● 絶対オフセットによる near ジャンプ

絶対オフセットによる near ジャンプは，レジスタあるいはメモリ上にある飛び先のアドレス（オフセットのみ）を使用してジャンプするものです．これを「絶対間接 near ジャンプ」と呼びます．

飛び先のアドレスは，オフセットのみなので near ポインタとして指定します．そのため，16ビットプログラムでは16ビットの near ポインタ，32ビットプログラムでは32ビットの near ポインタを使います．ジャンプのときは，この near ポインタの値がそのままレジスタ EIP にロードされます．このとき，16ビット near ポインタだった場合，レジスタ EIP の上位ワードにはゼロが入り，レジスタ EIP の下位ワード（レジスタ IP）に16ビット near ポインタがロードされます（**図2**）．

現在使われている Windows や Linux のプログラムは，すべて32ビットのプログラムなので，Windows や Linux のプログラムでは32ビット near ポインタを使うことになります．

● far ジャンプ

JMP 命令は，far ポインタで飛び先を指定することで，異なる物理セグメントへジャンプすることができます．

far ジャンプでは，レジスタ（E）IP のほかに，セグメントレジスタ CS も更新されます．far ポインタは，機械語命令に直接埋

〔リスト1〕MASMのNOP，JMP，Jcc，LOOP命令の記述例

```
                                  .586
                                  .model flat

00000000                          .data

00000000 00000004                 dtDWord dd      4

00000000                          .code
                                          org     0
00000000                          skip00:
00000000  90                              nop
00000001  EB FD                           jmp     skip00
00000003  EB 01                           jmp     skip01
00000005  90                              nop
00000006  90                      s#kip01: nop
00000007  90                              nop
00000008                          @@:
00000008  90                              nop
00000009  EB FD                           jmp     @b
0000000B  EB 01                           jmp     @f
0000000D  90                              nop
0000000E  90                      @@:     nop
                                  ;----------------------------
0000000F  E9 000000EC                     jmp     skip03

                                          org     100h
00000100                          skip03:
00000100  90                              nop
                                  ;----------------------------
00000101  FF E3                           jmp     ebx
00000103  FF 25 00000000 R                jmp     dtDWord
                                  ;============================
                                          org     1000h
00001000                          skip10:
00001000  90                              nop
00001001  74 FD                           jz      skip10
00001003  74 01                           jz      skip11
00001005  90                              nop
00001006  90                      skip11: nop
00001007  90                              nop
00001008                          @@:
00001008  90                              nop
00001009  74 FD                           jz      @b
0000100B  74 01                           jz      @f
0000100D  90                              nop
0000100E  90                      @@:     nop
                                  ;----------------------------
0000100F  0F 84 000000EB                  jz      skip12

                                          org     1100h
00001100                          skip12:
00001100  90                              nop
                                  ;============================
00001101  67& E3 FC                       jcxz    skip12
00001104  E3 FA                           jecxz   skip12
                                  ;============================
                                          org     2000h
00002000                          skip20:
00002000  90                              nop
00002001  90                              nop
00002002  90                              nop
00002003  E2 FB                           loop    skip20
00002005  67& E2 F8                       loopw   skip20
00002008  E2 F6                           loopd   skip20
                                  ;
0000200A  E1 F4                           loope   skip20
0000200C  E1 F2                           loopz   skip20
0000200E  67& E1 EF                       loopew  skip20
00002011  67& E1 EC                       loopzw  skip20
00002014  E1 EA                           looped  skip20
00002016  E1 E8                           loopzd  skip20
                                  ;
00002018  E0 E6                           loopne  skip20
0000201A  E0 E4                           loopnz  skip20
0000201C  67& E0 E1                       loopnew skip20
0000201F  67& E0 DE                       loopnzw skip20
00002022  E0 DC                           looped  skip20
00002024  E0 DA                           loopnzd skip20

                                  end
```


め込む「絶対farジャンプ」と，メモリで指定する「絶対間接farジャンプ」があります．また，このfarポインタによるジャンプは，セグメント外ジャンプのほかに，異なるタスクへの移行や特権レベルの異なるルーチンへのジャンプなどに使用されます．

本連載の第13回(2002年12月号)の最初でも述べたように，WindowsやLinuxの32ビットプログラムは，一つ大きなセグメントにコードもデータ(スタックを含む)も一緒にロードするため，一般のWindowsやLinuxのプログラムでは，セグメント外ジャンプはありません．

また，異なるタスクへの移行や特権レベルの異なるルーチンへのジャンプといったことは，OSやライブラリが行ってくれるため，一般のアプリケーションプログラムでは，このfarポインタによるジャンプは通常では使用することはありません．

● 実際のMASMおよびgasでのジャンプ命令の記述例

リスト1が，実際のMASMでのジャンプ命令(JMP，Jcc)の記述例を示したものです．そして**リスト2**に実際のgasでのジャンプ命令(JMP，Jcc)の記述例を示しています．

リスト1，**リスト2**とも，通常WindowsやLinuxの32ビット

〔リスト2〕gasのNOP，JMP，Jcc，LOOP命令の記述例

```
 1                          .data
 2
 3  0000 04000000           dtDWord:  .long  4
 4
 5                          .text
 6                                  .org    0
 7                          skip00:
 8  0000 90                         nop
 9  0001 EBFD                       jmp     skip00
10  0003 EB01                       jmp     skip01
11  0005 90                         nop
12  0006 90                 skip01: nop
13  0007 90                         nop
14                          1:
15  0008 90                         nop
16  0009 EBFD                       jmp     1b
17  000b EB01                       jmp     1f
18  000d 90                         nop
19  000e 90                 1:      nop
20                          #----------------------------
21  000f E9EC0000                   jmp     skip03
21       00
22
23  0014 00000000                   .org    0x100
23       00000000
23       00000000
23       00000000
23       00000000
24                          skip03:
25  0100 90                         nop
26                          #----------------------------
27  0101 FFE3                       jmp     *%ebx
28  0103 FF250000                   jmp     *dtDWord
28       0000
29                          #============================
30  0109 00000000                   .org    0x1000
30       00000000
30       00000000
30       00000000
30       00000000
31                          skip10:
32  1000 90                         nop
33  1001 74FD                       jz      skip10
34  1003 7401                       jz      skip11
35  1005 90                         nop
36  1006 90                 skip11: nop
37  1007 90                         nop
38                          1:
39  1008 90                         nop
40  1009 74FD                       jz      1b
41  100b 7401                       jz      1f
42  100d 90                         nop
43  100e 90                 1:      nop
44                          #----------------------------
45  100f 0F84EB00                   jz      skip12
45       0000
46
47  1015 00000000                   .org    0x1100
47       00000000
47       00000000
47       00000000
47       00000000
48                          skip12:
49  1100 90                         nop
50                          #============================
51  1101 67E3FC                     jcxz    skip12
52  1104 E3FA                       jecxz   skip12
53                          #============================
54  1106 00000000                   .org    0x2000
54       00000000
54       00000000
54       00000000
54       00000000
55                          skip20:
56  2000 90                         nop
57  2001 90                         nop
58  2002 90                         nop
59  2003 E2FB                       loop    skip20
60  2005 67E2F8                     loopw   skip20
61  2008 E2F6                       loopl   skip20
62                          ;
63  200a E1F4                       loope   skip20
64  200c E1F2                       loopz   skip20
65  200e 67E1EF                     loopew  skip20
66  2011 67E1EC                     loopzw  skip20
67  2014 E1EA                       loopel  skip20
68  2016 E1E8                       loopzl  skip20
69                          ;
70  2018 E0E6                       loopne  skip20
71  201a E0E4                       loopnz  skip20
72  201c 67E0E1                     loopnew skip20
73  201f 67E0DE                     loopnzw skip20
74  2022 E0DC                       loopnel skip20
75  2024 E0DA                       loopnzl skip20
```

プログラムでは使用されることがないfarジャンプについては割愛しました．そのため，**リスト1**，**リスト2**では，使用されることが多いnearジャンプのみを示しました．

ループ命令(LOOP，LOOPcc)

ループ命令のニモニックは，単純なループの命令が「LOOP」，状態付きのループ命令が「LOOPcc」です．

LOOPccのccの部分には「Z」，「E」，「NZ」，「NE」の条件を表す文字が入ります．条件「Z」，「E」，「NZ」，「NE」の意味は，**表1**のJccと同じです．機械語命令的にはLOOPZとLOOPEは同じコードで，LOOPNZとLOOPNEも同じコードとなっています．

ループ命令での飛び先のアドレスは，同じ物理セグメント内のみの「相対オフセット」で指定します．ただし，ループ命令にはshortジャンプしかないため，相対オフセットの値は－128～＋127の範囲を超えないようにする必要があります．

ループ命令は，ループ回数を数えるカウンタとして，レジスタCXあるいはレジスタECXを使用します．LOOP命令では，まずレジスタ(E)CXがデクリメント(－1)されます．その結果，レジスタ(E)CXがゼロでなければ，指定アドレスの命令にジャンプします．しかし，レジスタ(E)CXがゼロだった場合は，プログラムの順番どおりにLOOP命令の次の命令を実行します〔**図3(a)**〕．

LOOPcc命令の場合は，レジスタ(E)CXがゼロか否か調べられるところで，条件ccも調べられます〔**図3(b)**〕．

つまり，LOOPZおよびLOOPEは，レジスタ(E)CX ≠ 0でかつZF = 1(ゼロあるいは等しい)ならジャンプします．逆に，レジスタ(E)CX = 0あるいはZF = 0なら次の命令を実行します．また，LOOPNZおよびLOOPNEは，レジスタ(E)CX ≠ 0でかつZF = 0(ゼロでないあるいは等しくない)ならジャンプします．逆に，レジスタ(E)CX = 0あるいはZF = 1なら次の命令を実行します．

リスト1が実際のMASMでのループ命令の記述例，**リスト2**がgasでのループ命令の記述例です．

コール命令(CALL)

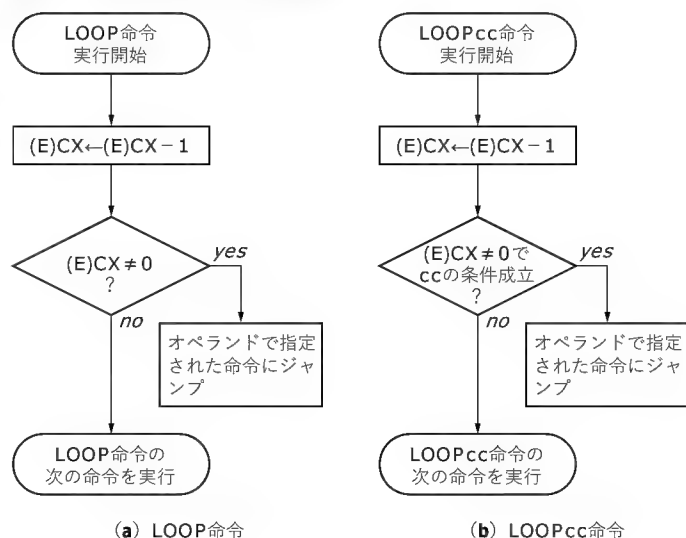
サブルーチンを呼び出す場合にはCALLを使用します．CALL命令は，無条件コールのみです．

CALL命令には，呼び出し先のアドレスの違いから，同じ物理セグメント内なら「nearコール」，異なる物理セグメントなら「farコール」を使うことになります．また，nearコールはさらにアドレスの指定方法の違いから「相対オフセット」，「絶対オフセット」の2種類があります．

● 相対オフセットによるnearコール

JMP命令で使われる相対オフセットと同じです．ただし，CALL命令にはshortタイプのコールはありません．セグメント内の全域をコール対象とした，相対オフセットのnearコール

〔図3〕LOOP，LOOPcc命令の動作

(a) LOOP命令　(b) LOOPcc命令

(相対nearコール)のみが使用できます．

● 絶対オフセットによるnearコール

JMP命令で使われる絶対オフセットと同じで，レジスタあるいはメモリ上にある呼び出し先のオフセット(nearポインタ)を使用してサブルーチンをコールするものです(絶対間接nearコール)．

● farコール

JMP命令と同じように，farポインタで呼び出し先を指定することで，異なる物理セグメントのサブルーチンをコールすることができます．

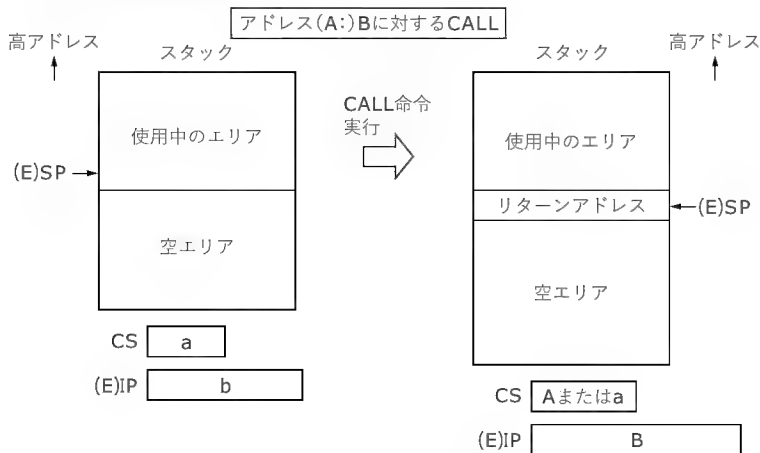
farコールでは，レジスタ(E)IPの他に，セグメントレジスタCSも更新されます．farコールで使われるfarポインタは，機械語命令に直接埋め込まれた「絶対farコール」と，メモリで指定する「絶対間接farコール」があります．また，このfarポインタによるコールは，セグメント外のサブルーチン呼び出しのほかに，異なるタスクにあるルーチンの呼び出しや特権レベルの異なるルーチンの呼び出しなどに使用されます．

JMP命令のところでも述べたように，WindowsやLinuxの32ビットプログラムでは，一つ大きなセグメントにコードもデータ(スタックを含む)も一緒にロードするため，一般のWindowsやLinuxのアプリケーションプログラムでは，セグメント外コールは使用しません．

また，異なるタスクにあるルーチンの呼び出しや特権レベルの異なるルーチンの呼び出しといったことは，OSやライブラリが行ってくれるため，このfarポインタによる呼び出しは通常使用することはありません．

● CALL命令の動作

CALL命令は，指定アドレスにジャンプする前に，サブルーチンから戻ってくるためのリターンアドレスをスタックにPUSHした後，呼び出し先のアドレスをレジスタ(E)IP(farコールなら

〔図4〕**CALL命令の動作**

CSも)に設定します(**図4**).

スタックにPUSHされるリターンアドレスは，セグメント内呼び出しならnearポインタが使われ，プログラムの実行モードが16ビットなら16ビットのnearポインタ，32ビットなら32ビットのnearポインタが使われます．また，セグメント外呼び出しならfarポインタがリターンアドレスとしてスタックにPUSHされます．この場合も，プログラムの実行モードにより，16ビットなら32ビットのfarポインタ，32ビットなら48ビットのfarポインタが使われます.

リスト3がMASMでのCALL命令の記述例，**リスト4**がgasでのCALL命令の記述例です．**リスト3**，**リスト4**とも通常のWindowsやLinuxの32ビットプログラムでは使用されることがないfarコールについては割愛しました．そのため，**リスト3**，**リスト4**では，使用されることが多いnearコールのみを示しました.

〔リスト3〕
MASMのCALL，RET，INT，IRET，BOUND，ENTER，LEAVE命令の記述例

```
                                  .586
                                  .model flat

00000000                          .data

00000000 00000004                 dtDWord dd      4         ─ 下限

00000004 0005 000C                dtWul   dw      5,12  }←─ BOUND命令で使用する上下限の値
00000008 00000005                 dtDWul  dd      5,12  }
         0000000C                                 └── 上限

00000000                          .code
                                          org     0
00000000                          skip00:
00000000  90                              nop
00000001  E8 FFFFFFFA                     call    skip00
00000006  E8 00000001                     call    skip01
0000000B  90                              nop
0000000C  90                      skip01: nop
0000000D  90                              nop
0000000E                          @@:
0000000E  90                              nop
0000000F  E8 FFFFFFFA                     call    @b
00000014  E8 00000001                     call    @f
00000019  90                              nop
0000001A  90                      @@:     nop
                                  ;----------------------------
0000001B  E8 000000E0                     call    skip03

                                          org     100h
00000100                          skip03:
00000100  C3                              ret
00000101  C2 0006                         ret     6 ←── オペランドがあるRET命令
                                  ;----------------------------
00000104  FF D3                           call    ebx
00000106  FF 15 00000000 R                call    dtDWord
                                  ;============================
                                          org     1000h
00001000  CD 20                           int     20h
00001002  CC                              int     3 ←── INT.3命令のみ1バイトのコードとなる
00001003  CE                              into
00001004  66| CF                          iret ←── 16ビットのIRET命令
00001006  CF                              iretd ←── 32ビットのIRET命令
                                  ;============================
00001007  66| 62 15                       bound   dx,dtWul ─ メモリ上の上下限値
          00000004 R
0000100E  62 15 00000008 R                bound   edx,dtDWul ─ インデックスの入ったレジスタ
                                  ;============================
00001014  C8 0008 00                      enter   8,0 ←── ネスティングレベル〔0=ネストなし，1〜31=ネストのレベル(図7参照)〕
00001018  C9                              leave
                                  end          └── スタック上に確保するローカル変数のバイト数
```

リターン命令(RET)

RET命令は，CALL命令に対応する形で，nearコールされたサブルーチンからの戻り「nearリターン」と，farコールされたサブルーチンからの戻り「farリターン」の二つの命令をもっています．

● リターン命令の動作

nearリターンでは，スタックに保存されているnearポインタのリターンアドレスをスタックからPOPし，レジスタ(E)IPにロードすることで，サブルーチンを呼び出した元のルーチンへ戻ります．

farリターンでは，スタックに保存されているfarポインタのリターンアドレスをスタックからPOPし，レジスタ(E)IPおよびセグメントレジスタCSにロードすることで，サブルーチンを呼び出した元のルーチンへ戻ります．また，CALL命令で特権レベルの異なるルーチンを呼び出した場合，RET命令により元の特権レベルのルーチンに戻ることができます．

先に述べたように，WindowsやLinuxの32ビットプログラムでは，セグメント外コールや特権レベルの異なるルーチンの呼び出しといったことは，一般的なアプリケーションプログラムでは行わないため，RET命令でもこのようなfarリターンは通常使用されません．

● リターン命令のオペランド

RET命令には，near/farの両方のリターンに，16ビットイミディエイトのオペランド付きとなしの二つの命令があります．オペランドなしのリターン命令は，今説明した動作のみを行います．

オペランド付きのRET命令は，上記で説明した動作のほかに，リターンアドレスをスタックからPOPした後，オペランドで指定された16ビットイミディエイトの値をレジスタ(E)SPに加算します(**図5**)．この16ビットイミディエイトのレジスタ(E)SPへの加算というのは，サブルーチンを呼び出すときスタックに積まれた実引き数を削除するのに使用します．

リスト3がMASMでのRET命令の記述例，**リスト4**がgasでのRET命令の記述例です．**リスト3**，**リスト4**ともに，通常はWindowsやLinuxの32ビットプログラムでは使用されることがないfarリターンについては割愛しました．そのため，**リスト3**，**リスト4**では，使用されることの多いnearリターンのみを示しました．

割り込み(例外)を発生させる命令(INT，INT 3，INTO，BOUND)と割り込み(例外)からのリターン命令(IRET)

INT命令，INT　3命令，INTO命令，BOUND命令は，プログラムでソフトウェア割り込みや例外を発生させるための命令です．そして，IRET命令がハードウェア割り込み，ソフトウェア割り込み，例外の処理ルーチン(ハンドラ)からのリターンとなります．

INT命令のようなソフトウェア割り込みや例外，ハードウェア割り込みが発生するとスタックには，レジスタ(E)FLAGSの値とリターン先となるCS，(E)IPの値が**図6**のようにPUSHされます．

〔リスト4〕gasのCALL，RET，INT，IRET，BOUND，ENTER，LEAVE命令の記述例

```
1                      .data
2
3   0000 04000000      dtDWord:   .long  4
4
5   0004 05000C00      dtWul:     .word  5,12
6   0008 05000000      dtDWul:    .long  5,12
6        0C000000
7
8                      .text
9                              .org    0
10                     skip00:
11  0000 90                    nop
12  0001 E8FAFFFF              call    skip00
12       FF
13  0006 E8010000              call    skip01
13       00
14  000b 90                    nop
15  000c 90            skip01: nop
16  000d 90                    nop
17                     1:
18  000e 90                    nop
19  000f E8FAFFFF              call    1b
19       FF
20  0014 E8010000              call    1f
20       00
21  0019 90                    nop
22  001a 90            1:      nop
23                     #----------------------------
24  001b E8E00000              call    skip03
24       00
25
26  0020 00000000              .org    0x100
26       00000000
26       00000000
26       00000000
26       00000000
27                     skip03:
28  0100 C3                    ret
29  0101 C20600                ret     $6
30                     #----------------------------
31  0104 FFD3                  call    *%ebx
32  0106 FF150000              call    *dtDWord
32       0000
33                     #============================
34  010c 00000000              .org    0x1000
34       00000000
34       00000000
34       00000000
34       00000000
35  1000 CD20                  int     $0x20
36  1002 CC                    int     $3
37  1003 CE                    into
38  1004 CF                    iret
39  1005 66CF                  iretw
40  1007 CF                    iretl
41                     #============================
42  1008 66621504              boundw  %dx,dtWul
42       000000
43  100f 62150800              boundl  %edx,dtDWul
43       0000
44                     #============================
45  1015 C8080000              enter   $8,$0
46  1019 C9                    leave
```

〔図5〕RET命令の動作

(a) オペランドなしのRET命令　(b) オペランド付きRET命令

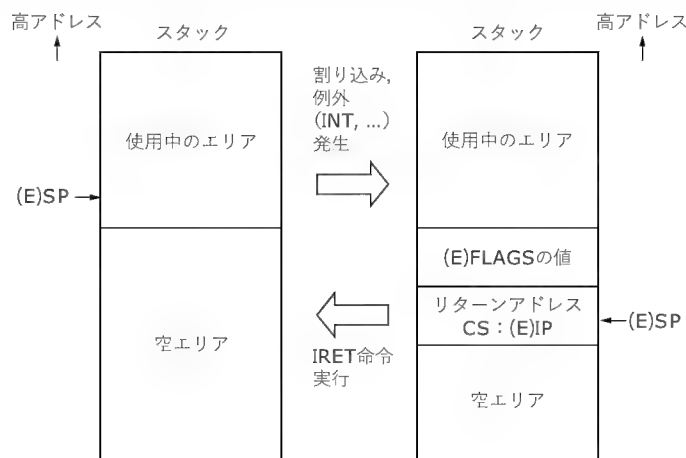
● INT命令，INT 3命令

INT命令は，8ビットイミディエイトのオペランドで指定された番号(0～255)のソフトウェア割り込みを発生させます．また，INT　3命令は，割り込み番号3のソフトウェア割り込みを発生させる命令です．通常INT命令は2バイト長なのですが，このINT　3命令のみ1バイト長となっています．

このINT　3命令は特別に用意された命令で，デバッガのブレークポインタとして使用するためのものです．そのため，命令長が1バイトになっています．

● INTO命令

INTO命令は，ステータスフラグのオーバフローフラグ(OF)が1になっていたとき，割り込み番号4の例外を発生させる命令です．x86系CPUは，演算でオーバフローした場合でも例外は発生せず，ステータスフラグがOF＝1となるだけです．そのため，オーバフローを例外で処理したい場合に，このINTO命令を使用します．ですから，オーバフローに対する例外処理が用意されている場合のみ，このINTO命令は使用できます．

〔図6〕INT，INT 3，INTO，BOUND命令，そしてIRET命令を実行したときのスタック

● BOUND命令

BOUND命令は，指定された配列のインデックスが有効範囲にあるか否かを調べるための命令です．インデックスはレジスタ，範囲指定の値はメモリに配置(最初の値が下限，次の値が上限)し，その値はともに16あるいは32ビットの符号付き整数で指定します．

もし，BOUND命令でインデックスが範囲外と判断されたら，割り込み番号5の例外が発生します．ただし，このBOUND命令は，INT，INT　3，INTO命令とは異なり，リターンアドレスは例外を発生させた次の命令ではなく，BOUND命令自体を指しているので，この点に注意が必要です．

このBOUND命令も，例外処理が用意されている場合にのみ使用できる命令です．

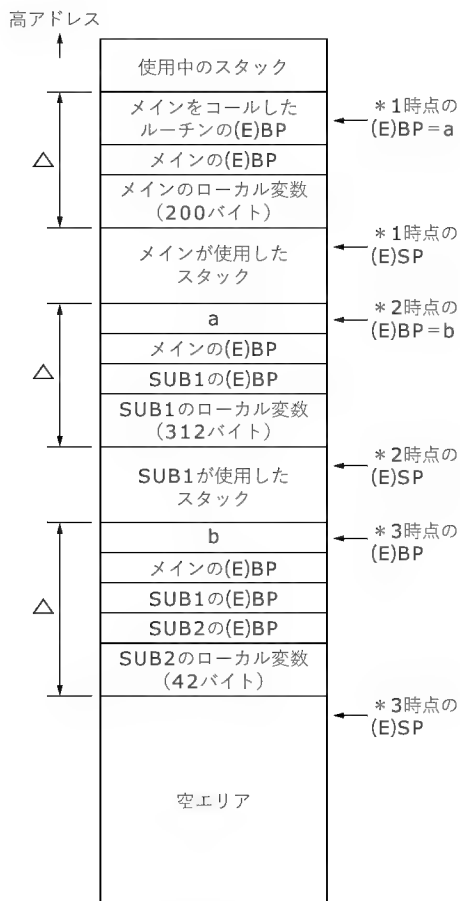
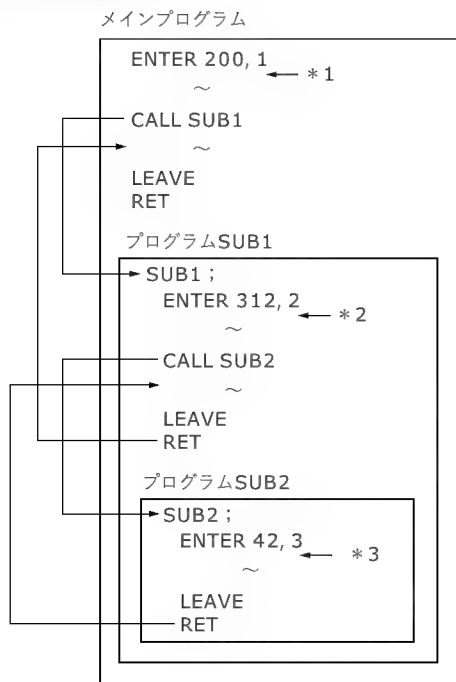
● IRET命令

IRET命令は，ハードウェア割り込みや例外，ソフトウェア割り込みの処理ルーチン(ハンドラ)からリターンするための専用の命令です．割り込みや例外が発生するとリターンアドレスに先立ち，レジスタ(E)FLAGSの値もスタックにPUSHされているので，通常のfarリターンによるRET命令は使用できません．そのため，割り込みや例外の処理からのリターンは，専用のIRET命令を使用します．IRET命令を実行すると，スタックは**図6**のように割り込み前の状態に戻ります．

● WindowsやLinuxではこれらの命令は使用できない

WindowsやLinuxでは，割り込みや例外のすべてをOSが管理しているため，一般のアプリケーションプログラムやドライバでは，勝手には割り込みや例外処理ルーチンを作ることはできません．そのため，ここで説明した割り込みや例外を発生させる命令は，アプリケーションなどのプログラムで使用することはありません．

また，WindowsやLinuxではドライバ上の割り込み処理ルーチン(ハンドラ)も，割り込みのエントリとリターンをOSが管理するため，ドライバ上でこのIRET命令を使用することもありません．

〔図7〕ENTER命令，LEAVE命令の動作

スタックフレームの作成と解放を行う命令（ENTER，LEAVE）

ENTER命令は，高級言語で使われるスタックフレームを作成します．そして，ENTER命令で作成したスタックフレームをLEAVE命令で解放します．そのため，ENTER命令はサブルーチンの最初で実行され，LEAVE命令はサブルーチンの最後のRET命令の直前に実行されます．

ENTER命令は二つのオペランドをもち，LEAVE命令にはオペランドがありません．ENTER命令の二つのオペランドは，ともにイミディエイトで，インテル表記で表すと，

```
ENTER   imm16, imm8
```

となります．

最初のimm16は，スタック上に確保するローカルな変数のための領域のバイトサイズです．このスタック上に確保されるローカルな変数とは，サブルーチン（手続きや関数）内でのみ使用され，サブルーチンから戻るとき消滅する変数のことです．C言語でいうauto記憶クラスの変数に相当します．

次のimm8は，ネスティングレベルを示すもので0～31のレベルを持ちます．このネスティングレベルは，PASCAL言語のような手続きの中に，さらにいくつもの手続きを入れ子（ネスト）の状態で記述できる言語で使用されます．このENTER命令の動作を図で表したのが**図7**です．

通常，アセンブラのプログラムでは，このような複雑な構造をしたサブルーチンは作成しません．つまり，ENTER命令は，CやPASCALなどのコンパイラが生成するオブジェクトとして使用されることを目的とした機械語命令だといえます．

LEAVE命令は，ENTER命令の作用を打ち消す動作として，

```
MOV   (E)SP, (E)BP
POP   (E)BP
```

の2命令に相当する動作をします．LEAVE命令自体は1バイトの命令であるため，この2命令を使う場面で，代わりにLEAVE命令を使うことでプログラムを短くすることができます．

＊　　　＊

次回は，x86系CPUの汎用命令の内，最後に残ったストリング命令の詳細と，CPU自体を制御するためのシステム命令の概略について説明する予定です．

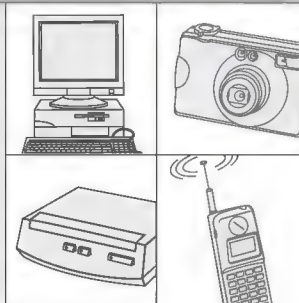
おおぬき・ひろゆき　大貫ソフトウェア設計事務所

家電機器をネットワーク化するアーキテクチャ
Universal Plug and Playの全貌

第2回 UPnPの規格概要(後編)

茶間 康

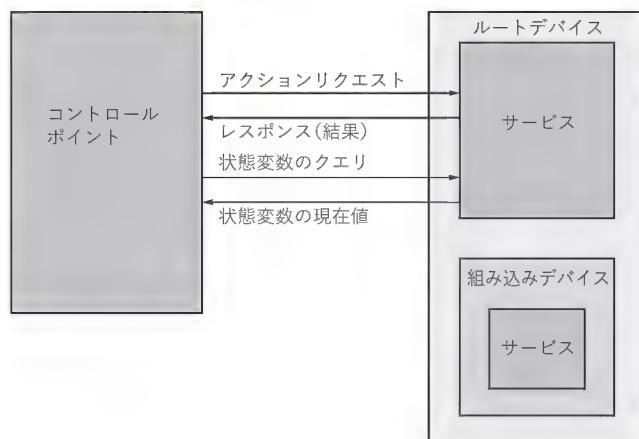
従来，ネットワーク機器を接続するためには，IPアドレスの設定や各種デバイスドライバのインストールなど，煩雑な設定が必要だった．これを解決する手段として提案されたのがUniversal Plug and Playである．UPnPにより，ネットワーク機器をPlug and Play感覚で使用することが可能になる．そして，その適応範囲はルータなどの純然たるネットワーク機器だけでなく，プリンタ，スキャナ，デジカメ，AV機器など，幅広い．また基盤技術として，SOAP(Simple Object Access Protocol)とXMLを使用しているため，理解が容易である．

そこで本稿では，注目度が高まるUPnPについて連載で解説し，その理解を深める．第1回と第2回(今回)ではUPnPの規格について解説を行い，第3回では実際のUPnPプログラミングを解説する．
(編集部)

第1回ではアドレッシング，ディスカバリ，ディスクリプションを説明しました．これでデバイスが何者で，どのような機能(サービス)をもっているかと，それを判別するしくみまでが理解できたと思います．

今回は，実際にデバイスを制御する方法であるコントロールと，それに付随するイベンティング，プレゼンテーションを解説します．

コントロール

コントロールは，UPnPネットワークにおいてステップ3になります．コントロールは文字どおりコントロールポイントがデバイスを制御することですが，実際にはサービスに対して要求を行います．つまりデバイスは，サービスのためのコンテナといえます．では，コントロールポイントはサービスに関してどのようなことができるのでしょうか？

1) アクションを実行すること(アクションの実行)
2) アクションの結果を得ること(アクションのレスポンス)
3) サービスのもっている変数の値を調べること(クエリの実行)
4) クエリの結果を得ること(クエリのレスポンス)

では，コントロールポイントがどのようにサービスに対して要求を行うかを説明します．ここでサービスディスクリプション(以降SerDescと記述)を思い出してください．コントロールのためのURLがあったと思います．ここへコントロールのためのメッセージを送ります．このメッセージに関しては後に説明します．デバイスは，ソースIPアドレスおよびポートにレスポンスメッセージを送信します．**図1**にコントロールの動作を示します．

〔図1〕コントロールの動作

UPnPのコントロールは，リモートプロシージャコール(RPC)技術の一つであるSOAPを使用します．SOAPはリモートプロシージャコールを実現するため，XML/HTTPの使用に関して定義しています．コントロールメッセージは，SOAPヘッダおよびボディエレメントをベースにフォーマットされ，HTTPを介してTCP/IPにより配信されます．また，デバイスは結果またはエラーをSOAPでカプセル化し，HTTPリクエストを介して送信します．コントロールポイントは，HTTPレスポンスを介して受信します．**図2**にコントロールで使用するプロトコルを示します．

- アクションの実行

SOAPは，追加のHTTPヘッダを定義しています．これらがHTTP拡張子と混同されないよう，HTTP Extension Frameworkにしたがって，MANヘッダにSOAP独自のURIお

〔図2〕コントロールにおいて使用するプロトコル

UPnPベンダ
UPnP Forum
UPnPデバイスアーキテクチャ
SOAP
HTTP
TCP
IP

〔**表1**〕**コントロールメッセージ**(POST)

```
POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"

<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>in arg value</argumentName>
  other in args and their values go here,if any
    </u:actionName>
  </s:Body>
</s:Envelope>
```

リクエストライン			
POST	HTTPにより定義されたメソッド		
	path control URL	このサービスのコントロールのためのURLのパスコンポーネント(デバイスディスクリプションのサービスエレメントのcontrolURLサブエレメント).単一の相対URL	
HTTP/1.1	HTTPバージョン		
ヘッダ			
HOST	必須項目.このサービスのコントロールのためのURLのドメイン名またはIPアドレスおよび任意のポートコンポーネント(デバイス記述のサービスエレメントのcontrolURLサブエレメント).ポートが空,あるいは明記されていない場合は,80が用いられる		
ACCEPT-LANGUAGE	(コントロールメッセージではACCEPT-LANGUAGEヘッダは使用されない)		
CONTENT-LENGTH	必須項目.バイト単位で示したボディの長さ.整数		
CONTENT-TYPE	必須項目.text/xmlでなくてはならない.使用される文字コーディング(例:utf-8)を含む		
MAN	(メソッドPOSTによるリクエストにはMANヘッダはない)		
SOAPACTION	SOAPが定義する必須ヘッダ.サービスタイプ,ハッシュ文字,そして呼び出すアクションで,すべてダブルクォーテーションで括られる.メソッドM-POSTによるリクエストで用いられる場合,ヘッダ名はMANヘッダで定義されるHTTPネームスペースにより許可されたものでなくてはならない.単一のURI		
ボディ			
Envelope	SOAPが定義する必須エレメント.xmlnsの名前属性は,必ず"http://schemas.xmlsoap.org/soap/envelope/"となる.また,必ず"http://schemas.xmlsoap.org/soap/envelope/"という値のencodingStyle属性を含む.以下のサブエレメントを含む		
	Body	SOAPが定義する必須エレメント.SOAPネームスペースにより認可されたもの.以下のサブエレメントを含む	
		actionName	必須項目.名前エレメントは呼び出すアクションの名前.xmlnsネームスペース属性は,ダブルクォーテーションで括られたサービスタイプ.ボディの最初のサブエレメントでなくてはならない.以下の順序でサブエレメントを含む
		argumentName	アクションにin引き数がある場合のみ必須項目.アクションに渡される値.各in引き数に対し1度繰り返す(ネームスペースにより認可されていないエレメント名.エレメントネスト化コンテンツで充分).UPnPサービスディスクリプションに定義される単一のデータ型

よび接頭辞M-のついたHTTPメソッドを指定します.この場合,メソッドはM-POSTとなります.M-POSTを使用する場合,HTTPサーバがSOAP独自のURIおよびSOAP固有のヘッダを見つけ,理解する必要があります.ファイアウォールやプロキシが柔軟な管理をできるようにするために,SOAPではリクエストするとき,最初にMANヘッダやM-接頭辞が必須となります.

リクエストが"05 Method Not Allowed"というレスポンスで拒否された場合には,次のリクエストはMANヘッダとM-接頭辞を用いて送信されます.そのリクエストが"01 Not Implemented"あるいは"10 Not Extended"というレスポンスで拒否された場合,リクエストは失敗となります(他のHTTPレスポンスは,HTTPの仕様にしたがって処理される).

以下に,POSTメソッド(MANヘッダなし)を用いて送信されるコントロールメッセージを**表1**に示します.それに続いて,M-POSTメソッドおよびMANヘッダを用いて送信されるコントロールメッセージを**表2**に示します.

● アクションのレスポンス

コントロールポイントからのアクションの実行要求に対して結果を返します.サービス(デバイス)はSOAPでカプセルし,HTTPリクエストを介して送信し,コントロールポイントはHTTPレスポンスを介して受信されます.サービスは,転送時間を含めアクションの実行とレスポンスを30秒以内に完了しなければなりません.これより長くかかるアクションは,すぐ

〔**表2**〕**コントロールメッセージ**(M-POST)

```
M-POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
MAN: "http://schemas.xmlsoap.org/soap/envelope/"; ns=01
```

(メソッドM-POSTのリクエストのメッセージボディは，メソッドPOSTのリクエストのボディと同様)

リクエストライン		
M-POST	HTTP Extension Frameworkにより定義されたメソッド	
	path of control URL	このサービスのコントロールのためのURLのパスコンポーネント(デバイスディスクリプションのサービスエレメントのcontrolURLサブエレメント)．単一の相対URL
	HTTP/1.1	HTTPバージョン
ヘッダ		
HOST	必須項目．このサービスのコントロールのためのURLのドメイン名またはIPアドレスおよび任意のポートコンポーネント(デバイスディスクリプションのサービスエレメントのcontrolURLサブエレメント)．ポートが空，あるいは明記されていない場合は80が用いられる	
ACCEPT-LANGUAGE	(コントロールメッセージではACCEPT-LANGUAGEヘッダは使用されない)	
CONTENT-LENGTH	必須項目．バイト単位で示したボディの長さ．整数	
CONTENT-TYPE	必須項目．text/xmlでなくてはならない．使用される文字コーディング(例：utf-8)を含む	
MAN	必須項目．必ず"http://schemas.xmlsoap.org/soap/envelope/"となる．ns命令は，他のSOAPヘッダ(例：SOAPACTION)に対しネームスペース(例：01)を定義する	
SOAPACTION	SOAPが定義する必須ヘッダ．サービスタイプ，ハッシュ文字，そして呼び出すアクションで，すべてダブルクォーテーションで括られる．メソッドM-POSTによるリクエストで用いられる場合，ヘッダ名はMANヘッダで定義されるHTTPネームスペースにより許可されたものでなくてはならない．単一のURI	

にいったんレスポンスし，完了時にイベンティングを送信することにより結果を伝えるようにする必要があります(イベンティングを参照)．

実行が成功したときのアクションレスポンスメッセージを**表3**に示します．また，サービスが，コントロールポイントからのアクションを実行中にエラーが発生した場合のアクションレスポンスメッセージを**表4**(pp.167-168)に示します．なお，定義されたエラータイプ(errorCodeおよびerrorDescription)エレメントに対応する値を**表5**に示します．

● クエリの実行

コントロールポイントがサービスへアクションを実行させることに加え，サービスへクエリメッセージを送信し，ステートバリアブル(状態変数)の値を調査することができます．一つのクエリメッセージで問い合わせることができるのは一つの状態変数のみです．複数の状態変数を問い合わせるためには，複数のクエリメッセージを送信する必要があります．コントロールポイントがステートバリアブル(状態変数)の値を問い合わせるためのメッセージの形式を**表6**(pp.168-169)に示します．

なお，POSTメソッドによるリクエストが"05 Method Not Allowed"という応答で拒否された場合，すでにアクションの実行で説明したように，コントロールポイントはMANヘッダとM-接頭辞を用いたリクエストを送信します．

● クエリのレスポンス

コントロールポイントからのステートバリアブル(状態変数)値の問い合わせにレスポンスするため，サービスは転送時間を含めて30秒以内に応答しなければなりません．サービスがこの時間内に応答できなかった場合にコントロールポイントが行うべきことは，アプリケーションにより異なります．クエリが成功したときのクエリレスポンスメッセージを**表7**(p.169)に示します．サービスがコントロールポイントからのクエリを実行中にエラーが発生した場合のクエリレスポンスメッセージを**表8**(p.170)に示します．

〔**表5**〕**コントロールのエラーコード**

errorCode	errorDescription	記　述
401	Invalid Action	このサービスには指定された値のアクションは存在しない
402	Invalid Args	以下のいずれかである可能性がある：in引き数が不十分，in引き数が過剰，指定された名前のin引き数が存在しない，一つ以上のin引き数のデータ型が不正
403	Out of Sync	同期していない
501	Action Failed	サービスの現在の状態によりそのアクションの呼び出しが不可能になっている可能性がある
600-699	TBD	共通アクションエラー．UPnP Forum WCにより定義
700-799	TBD	標準アクションのアクション固有エラー．UPnP Forum WCにより定義
800-899	TBD	非標準アクションのアクション固有エラー．UPnPベンダにより定義

〔表3〕アクションレスポンスメッセージ（サクセス）

```
HTTP/1.1 200 OK
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
EXT:
SERVER: OS/version UPnP/1.0 product/version

<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionNameResponse xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>out arg value</argumentName>
  other out args and their values go here,if any
    </u:actionNameResponse>
  </s:Body>
</s:Envelope>
```

リクエストライン				
HTTP/1.1	HTTPバージョン			
	200 OK	HTTPサクセスコード		
ヘッダ				
CONTENT-LANGUAGE	(コントロールメッセージではCONTENT-LANGUAGEヘッダは使用されない)			
CONTENT-LENGTH	必須項目．バイト単位で示したボディの長さ．整数			
CONTENT-TYPE	必須項目．text/xmlでなくてはならない．使用される文字コーディング(例：utf-8)を含む			
DATE	推奨項目．アクションのレスポンスメッセージが生成された日付を指定する．RFC 1123で定義されている日付と時間コードで指定する			
EXT	必須項目．MANヘッダの値を理解したかどうかの確認(ヘッダのみ．値なし)			
SERVER	必須項目．OS名，OSバージョン，UPnP/1.0，製品名および製品バージョンを連記したもの．文字列			
ボディ				
Envelope	SOAPが定義する必須エレメント．xmlnsの名前属性は，必ず"http://schemas.xmlsoap.org/soap/envelope/"となる．また，必ず"http://schemas.xmlsoap.org/soap/envelope/"という値のencodingStyle属性を含む．以下のサブエレメントを含む			
	Body	SOAPが定義する必須エレメント．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む		
		actionNameResponse	必須項目．レスポンスされたアクションの名前．xmlnsネームスペース属性は，ダブルクォーテーションで括られたサービスタイプ．ボディの最初のサブエレメントでなくてはならない．以下のサブエレメントを含む	
			argumentName	アクションにout引き数がある場合のみ必須項目．アクションから返される値．各out引き数に対し1度繰り返し．アクションがretvalとマークされた引き数をもつ場合，この引き数が最初のエレメントとなる(ネームスペースにより認可されていないエレメント名．エレメントネスト化コンテンツで充分)．UPnPサービス記述に定義される単一のデータ型

〔表4〕アクションレスポンスメッセージ（エラー）

```
HTTP/1.1 500 Internal Server Error
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
EXT:
SERVER: OS/version UPnP/1.0 product/version

<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
          <errorCode>error code</errorCode>
          <errorDescription>error string</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
```

〔表4〕**アクションレスポンスメッセージ**(エラー)(つづき)

```
  </s:Body>
</s:Envelope>
```

リクエストライン							
HTTP/1.1	HTTPバージョン						
	500 Internal Server Error	HTTPエラーコード					
ヘッダ							
CONTENT-LANGUAGE	(コントロールメッセージではCONTENT-LANGUAGEヘッダは使用されない)						
CONTENT-LENGTH	必須項目．バイト単位で示したボディの長さ．整数						
CONTENT-TYPE	必須項目．text/xmlでなくてはならない．使用される文字コーディング(例：utf-8)を含む						
DATE	推奨項目．アクションのレスポンスメッセージが生成された日付を指定する．RFC 1123で定義されている日付と時間コードで指定する						
EXT	必須項目．MANヘッダが理解されたかどうかの確認(ヘッダのみ．値なし)						
SERVER	必須項目．OS名，OSバージョン，UPnP/1.0，製品名および製品バージョンを連記したもの．文字列						
ボディ							
Envelope	SOAPが定義する必須エレメント．xmlnsの名前属性は，必ず"http://schemas.xmlsoap.org/soap/envelope/"となる．また，必ず"http://schemas.xmlsoap.org/soap/envelope/"という値のencodingStyle属性を含む．以下のサブエレメントを含む						
	Body	SOAPが定義する必須エレメント．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む					
		Fault	SOAPが定義する必須エレメント．アクション実行中にエラーが発生したことを示す．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む				
			faultcode	SOAPが定義する必須エレメント．値はSOAPネームスペースにより認可されたものでなくてはならない．Clientでなくてはならない			
			faultstring	SOAPが定義する必須エレメント．UPnPErrorでなくてはならない			
			detail	SOAPが定義する必須エレメント			
				UPnPError	UPnPが定義する必須エレメント		
					errorCode	UPnPが定義する必須エレメント．どのエラーが発生したかを識別するコード．表参照．整数	
					errorDescription	UPnPが定義する推奨エレメント．短い説明．**表5**参照．文字列．256文字未満であることが推奨される	

〔表6〕**ステートバリアブルの値を問い合わせるためのメッセージ**

```
POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-org:control-1-0#QueryStateVariable"

<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:QueryStateVariable xmlns:u="urn:schemas-upnp-org:control-1-0">
      <u:varName>variableName</u:varName>
    </u:QueryStateVariable>
  </s:Body>
<s:Envelope>
```

リクエストライン			
POST	HTTPにより定義されたメソッド		
	path of control URL	このサービスのコントロールのためのURLのパスコンポーネント(デバイスディスクリプションのサービスエレメントのcontrolURLサブエレメント)．単一の相対URL	
	HTTP/1.1	HTTPバージョン	
ヘッダ			
HOST	必須項目．このサービスのコントロールのためのURLのドメイン名またはIPアドレスおよび任意のポートコンポーネント(デバイスディスクリプションのサービスエレメントのcontrolURLサブエレメント)．ポートが空の場合，あるいは明記されていない場合，80が用いられる		
ACCEPT-LANGUAGE	(コントロールメッセージではACCEPT-LANGUAGEヘッダは使用されない)		
CONTENT-LENGTH	必須項目．バイト単位で示したボディの長さ．整数		
CONTENT-TYPE	必須項目．text/xmlでなくてはならない．使用される文字コーディング(例：utf-8)を含む		
MAN	(メソッドPOSTによるリクエストにはMANヘッダはない)		

〔表6〕ステートバリアブルの値を問い合わせるためのメッセージ(つづき)

<table>
<tr><td>SOAPACTION</td><td colspan="4">SOAPが定義する必須ヘッダ．“urn: schemas-UPnP-org: control-1-0#QueryStateVariable”でなくてはならない．メソッドM-POSTによるリクエストで用いられる場合，ヘッダ名はMANヘッダで定義されるHTTPネームスペースにより許可されたものでなくてはならない．単一のURI</td></tr>
<tr><td colspan="5">ボディ</td></tr>
<tr><td>Envelope</td><td colspan="4">SOAPが定義する必須エレメント．xmlnsの名前属性は，必ず“http://schemas.xmlsoap.org/soap/envelope/”となる．また，必ず“http://schemas.xmlsoap.org/soap/envelope/”という値のencodingStyle属性を含む．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="4"></td><td rowspan="4">Body</td><td colspan="3">SOAPが定義する必須エレメント．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="2">QueryStateVariable</td><td colspan="2">UPnPが定義する必須エレメント．アクション名．xmlnsネームスペース属性は，必ず“urn:schemas-UPnP-org:control-1-0”になる．Bodyの最初のサブエレメントでなくてはならない．以下の順序でサブエレメントを含む</td></tr>
<tr><td>varName</td><td>UPnPが定義する必須エレメント．変数名</td></tr>
<tr><td>QueryStateVariable</td><td colspan="2">ネームスペースにより認可されたもの．値は，問い合わせる状態変数の名前となる．文字列</td></tr>
</table>

〔表7〕クエリレスポンスメッセージ(サクセス)

```
HTTP/1.1 200 OK
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
EXT:
SERVER: OS/version UPnP/1.0 product/version

<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:QueryStateVariableResponse xmlns:u="urn:schemas-upnp-org:control-1-0">
      <return>variable value</return>
    </u:QueryStateVariableResponse>
  </s:Body>
</s:Envelope>
```

<table>
<tr><td colspan="5">リクエストライン</td></tr>
<tr><td rowspan="2">HTTP/1.1</td><td colspan="4">HTTPバージョン</td></tr>
<tr><td>200 OK</td><td colspan="3">HTTPサクセスコード</td></tr>
<tr><td colspan="5">ヘッダ</td></tr>
<tr><td>CONTENT-LANGUAGE</td><td colspan="4">(コントロールメッセージではCONTENT-LANGUAGEヘッダは使用されない)</td></tr>
<tr><td>CONTENT-LENGTH</td><td colspan="4">必須項目．バイト単位で示したボディの長さ．整数</td></tr>
<tr><td>CONTENT-TYPE</td><td colspan="4">必須項目．text/xmlでなくてはならない．使用される文字コーディング(例：utf-8)を含む</td></tr>
<tr><td>DATE</td><td colspan="4">推奨項目．クエリのレスポンスメッセージが生成された日付を指定する．RFC 1123で定義されている日付と時間コードで指定する</td></tr>
<tr><td>EXT</td><td colspan="4">必須項目．MANヘッダが理解されたかどうかの確認(ヘッダのみ．値なし)</td></tr>
<tr><td>SERVER</td><td colspan="4">必須項目．OS名，OSバージョン，UPnP/1.0，製品名および製品バージョンを連記する．文字列</td></tr>
<tr><td colspan="5">ボディ</td></tr>
<tr><td rowspan="4">Envelope</td><td colspan="4">SOAPが定義する必須エレメント．xmlnsの名前属性は，必ず“http://schemas.xmlsoap.org/soap/envelope/”となる．また，必ず“http://schemas.xmlsoap.org/soap/envelope/”という値のencodingStyle属性を含む．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="3">Body</td><td colspan="3">SOAPが定義する必須エレメント．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="2">QueryStateVariableResponse</td><td colspan="2">UPnPおよびSOAPが定義する必須エレメント．xmlnsネームスペース属性は必ず“urn:schemas-UPnP-org: control-1-0”になる．Bodyの最初のサブエレメントでなくてはならない．以下のサブエレメントを含む</td></tr>
<tr><td>return</td><td>UPnPが定義する必須エレメント．(ネームスペースにより認可されていないエレメント名．エレメントネスト化コンテンツで充分)．値は，リクエストのvarNameエレメントで指定された状態変数の現在の値</td></tr>
</table>

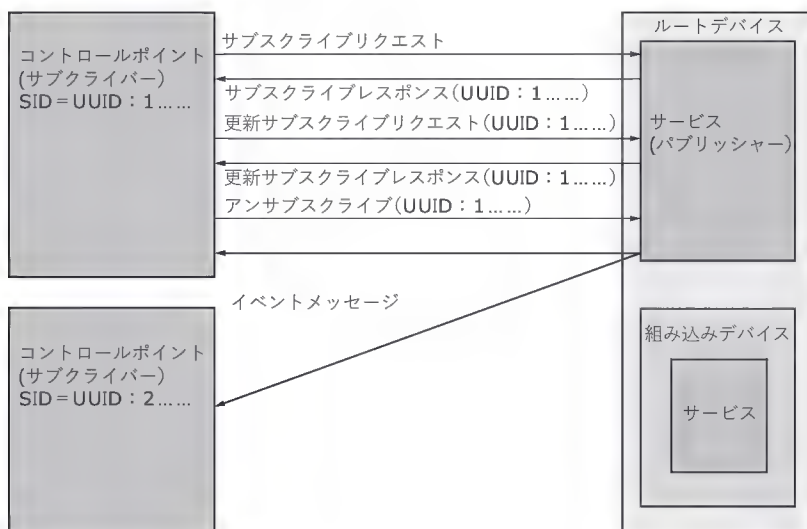
〔**表8**〕**クエリレスポンスメッセージ**(エラー)

```
HTTP/1.1 500 Internal Server Error
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
EXT:
SERVER: OS/version UPnP/1.0 product/version

<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
          <errorCode>error code</errorCode>
          <errorDescription>error string</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

<table>
<tr><td colspan="7">リクエストライン</td></tr>
<tr><td rowspan="2">HTTP/1.1</td><td colspan="6">HTTPバージョン</td></tr>
<tr><td colspan="3">500 Internal Server Error</td><td colspan="3">HTTPエラーコード</td></tr>
<tr><td colspan="7">ヘッダ</td></tr>
<tr><td>CONTENT-LANGUAGE</td><td colspan="6">(コントロールメッセージではCONTENT-LANGUAGEヘッダは使用されない)</td></tr>
<tr><td>CONTENT-LENGTH</td><td colspan="6">必須項目．バイト単位で示したボディの長さ．整数</td></tr>
<tr><td>CONTENT-TYPE</td><td colspan="6">必須項目．text/xmlでなくてはならない．使用される文字コーディング(例：utf-8)を含む</td></tr>
<tr><td>DATE</td><td colspan="6">推奨項目．クエリのレスポンスメッセージが生成された日付を指定する．RFC 1123で定義されている日付と時間コードで指定する</td></tr>
<tr><td>EXT</td><td colspan="6">必須項目．MANヘッダが理解されたかどうかの確認(ヘッダのみ．値なし)</td></tr>
<tr><td>SERVER</td><td colspan="6">必須項目．OS名，OSバージョン，UPnP/1.0，製品名および製品バージョンを連記したもの．文字列</td></tr>
<tr><td colspan="7">ボディ</td></tr>
<tr><td>Envelope</td><td colspan="6">SOAPが定義する必須エレメント．xmlnsの名前属性は，必ず“http://schemas.xmlsoap.org/soap/envelope/”となる．また，必ず“http://schemas.xmlsoap.org/soap/encoding/”という値のencodingStyle属性を含む．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="8"></td><td rowspan="8">Body</td><td colspan="5">SOAPが定義する必須エレメント．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="7">Fault</td><td colspan="4">SOAPが定義する必須エレメント．サービスが変数の値を返せなかった理由．SOAPネームスペースにより認可されたもの．以下のサブエレメントを含む</td></tr>
<tr><td>faultcode</td><td colspan="3">SOAPが定義する必須エレメント．値はSOAPネームスペースにより認可されたもの．Clientでなくてはならない</td></tr>
<tr><td>faultstring</td><td colspan="3">SOAPが定義する必須エレメント．errorCodeを記述する包括的なUPnP文字列．表15参照</td></tr>
<tr><td rowspan="4">detail</td><td colspan="3">SOAPが定義する必須エレメント．以下のサブエレメントを含む</td></tr>
<tr><td rowspan="3">UPnPError</td><td colspan="2">UPnPが定義する必須エレメント．以下のサブエレメントを含む</td></tr>
<tr><td>errorCode</td><td>UPnPが定義する必須エレメント．どのエラーが発生したかを識別するコード．表17参照．整数</td></tr>
<tr><td>errorDescription</td><td>UPnPが定義する推奨エレメント．短い説明．表9参照．文字列．256文字未満であることが推奨される</td></tr>
</table>

〔**表9**〕**クエリのエラーコード**

errorCode	errorDescription	記　述
404	Invalid Var	このサービスには指定された値の状態変数は存在しない
600-624	TBD	共通アクションエラー．UPnP Forum WCにより定義
625-649	TBD	将来の使用のためリザーブされている
650-674	TBD	標準アクションのアクション固有エラー．UPnP Forum WCにより定義
675-699	TBD	非標準アクションのアクション固有エラー．UPnPベンダにより定義

〔図3〕イベンティングの動作

なお，定義されたエラータイプ(errorCodeおよびerrorDescription)エレメントに対応する値を**表9**に示します．

イベンティング

イベンティングは，UPnPネットワークではステップ4になります．コントロールポイントがディスカバリ(ステップ1)，デバイスとサービスのディスクリプション(ステップ2)を終了した時点で，コントロールポイントはイベンティングの処理に移ることができます．コントロール(ステップ3)はイベンティングと連携しますが，実行されている必要はありません．

では，イベンティングは，どのような状況で有効なのかを解説します．コントロールポイントがアクションを実行した場合，当然コントロールポイントは結果を知ることができます．しかし，アクションの実行とは関係なくデバイスの状況が変わる場合はどうでしょうか？　たとえば，“プリンタのインクや紙が切れた”や“セキュリティシステムが異常を検知した”というような場合です．この場合，何らかの通知が必要になります．

イベンティングは，変化があったときすべてのコントロールポイントへマルチキャストで通知するわけではありません．コントロールポイントは，あらかじめチェックしておきたいデバイスのサービスへ通知を要求します．これをサブスクライブ(以降，購読と記述)と呼びます．サービス(イベンティングでは，このときのサービスをパブリッシャーと呼ぶ)は要求したコントロールポイント(購読するコントロールポイントをイベンティングではサブスクライバーと呼ぶ)へだけ，個別にユニキャストで通知します．サブスクライバー(以降，購読者と記述)およびパブリッシャー(以降，発行者と記述)がイベントメッセージに関して行うことを説明します．

1) サブスクライブリクエスト

購読者が発行者へイベントメッセージを受け取るためのリクエストメッセージを送信します．

2) サブスクライブレスポンス

発行者は，購読を許可するためのレスポンスメッセージを送信し購読者からの要求を登録します．このとき購読のタイムアウト時間をメッセージ内に示します．この時間が過ぎるまでに，購読者はタイムアウト時間を更新するため，再度サブスクライブリクエストメッセージを送信する必要があります．新聞を購読しているのに1か月ごとに要求しないと新聞配達が止まってしまうというようなしくみです(日本の場合は断らないと新聞配達は止まらないが)．

UPnPでも明示的に断りたい場合があります．それが3)アンサブクライブです．

3) アンサブスクライブ

購読者は登録された要求をキャンセルしたいときがあります．購読者は発行者へアンサブスクライブメッセージを送信し，キャンセルします．

4) アンサブスクライブレスポンス

発行者はキャンセルされたことを知らせるためのレスポンスメッセージを送信します．

5) イベンティング

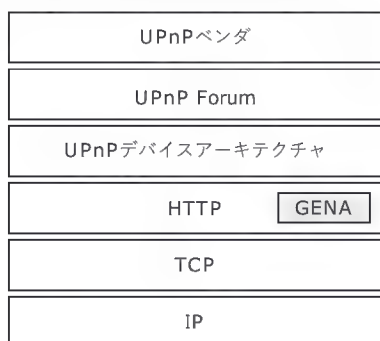
発行者はサービスの状態変数が変化した場合，イベントメッセージを購読者へ送信します．

6) イベンティングレスポンス

購読者は，イベントメッセージを受信したことを知らせるためレスポンスメッセージを送信します．

以上までのイベンティングの動作のまとめを**図3**に，イベンティングにおいて使用するプロトコルを**図4**に示します．

イベントメッセージは，GENAメソッド/ヘッダを用いて

〔図4〕イベンティングにおいて使用するプロトコル

フォーマットされ，HTTPを介してTCP/IPにより配信されます．

もう少しイベンティングについて説明します．購読者が発行者へ“購読する”と伝えると，発行者は最初に特殊なイベントメッセージを発行します．これをイニシャルイベントメッセージといいます．このメッセージにはイベントとして通知しなければならない変数の名前と，その時点でのすべて変数の値が含まれます．

購読者は，この情報を初期値として保存する必要があります．なぜ，このようなことをするかというと，イベントメッセージを変数単位で購読するためのメカニズムがないからです．そのため購読時には，すべての変数の情報が発行者から送られます．購読者は，初期値と比較して各変数が変化したことを認識します．

発行者は，各購読者からのサブスクライブリクエストを管理するため四つの情報をもったリスト（サブスクライバーリスト）を管理する必要があります．

1）SID（subscription identifier）

特定の購読を管理するための識別子．普遍的な一意の値でなくてはなりません．発行者はサブスクライブレスポンスのメッセージにこれをセットし，購読者がリクエストした購読を識別できるようにします．

2）delivery URL for event messages

購読者がイベントメッセージを受け取れるURLを，サブスクライブメッセージに含めて送信します．発行者はこれを保存し，イベントメッセージを送信しなければならないときに参照します．

3）event key

このキーはカウンタになっています．イニシャルイベントメッセージの場合，このキーは0となります．イベントメッセージを送信するごとにシーケンシャルにインクリメントします．連続した番号を受け取ることで，購読者はイベントメッセージに損失がないことを確認できます．

4）subscription duration

購読の有効時間です．この時間が過ぎる前に更新のサブスクライブメッセージを受け取らなければ，発行者は，この購読を無効にします．

これらの情報によりイベントメッセージを制御します．

次にタイムアウトに関して説明します．UPnPネットワークでは，購読の有効時間を決めることが可能です．これは，ディスカバリのCACHE-CONTROLと同じ時間が良いでしょう．購読者は，この時間内に発行者へ更新のサブスクライブメッセージを送信します．更新することでリスト内の購読が有効であるこ

〔表10〕サブスクライブメッセージ

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host: publisher port
<SPAN class=gena>CALLBACK: <delivery URL>
NT: UPnP:event
TIMEOUT: Second-requested subscription duration
<BR>
```

（サブスクライブメソッドのリクエストにはボディはないが，メッセージの最後にブランクラインが必要）

リクエストライン		
SUBSCRIBE	GENAにより定義されたメソッド	
	publisher path	イベンティングURL（デバイスディスクリプションのサービスエレメントのeventSubURLサブエレメント）のパスコンポーネント．単一の相対URL
	HTTP/1.1	HTTPバージョン
ヘッダ		
HOST	必須項目．イベントURL（デバイスディスクリプションのサービスエレメントのeventSubURLサブエレメント）のドメイン名またはIPアドレスおよび任意のポートコンポーネント．ポートが空の場合，あるいは明記されていない場合，80が用いられる	
CALLBACK	GENAが定義する必須ヘッダ．イベントメッセージの送信先．UPnPベンダにより定義．複数のURLが存在する場合，サービスがイベントを送信する際，いずれかが成功するまでこれらのURLを順に試す．一つ以上のURLは，アングルブラケットで区切られる	
NT	GENAが定義する必須ヘッダ．通知タイプ（Notification Type）のこと．UPnP:eventでなくてはならない	
TIMEOUT	推奨項目．サブスクライブが期限切れになるまでの期間．秒数または無限．UPnP Forum WCにより推奨される．UPnPベンダにより定義．キーワードSecond-の後に整数値が続く（スペースなし），あるいはinfinite（無限）	

〔表11〕サブスクライブメッセージレスポンスで発生するエラー

Incompatible headers
400 Bad Request．SIDヘッダおよびNTまたはCALLBACKヘッダのいずれかが存在する場合，発行者はHTTPエラー400 Bad Requestで応答する
Missing or invalid CALLBACK
412 Precondition Failed．CALLBACKヘッダがない場合，あるいは有効なHTTP URLを含んでいない場合，発行者はHTTPエラー412 Precondition Failedで応答する
Invalid NT
412 Precondition Failed．NTヘッダがUPnP:eventでない場合，発行者はHTTPエラー412 Precondition Failedで応答する
Unable to accept subscription
5xx 発行者が購読を許可できない場合，HTTP 500-シリーズのエラーコードで応答する

〔表12〕サブスクライブレスポンスメッセージ

```
HTTP/1.1 200 OK
DATE: when response was generated
SERVER: OS/version UPnP/1.0 product/version
<SPAN class=gena>SID: uuid: subscription-UUID
T IMEOUT: Second-actual subscription duration
<BR>
```

（サブスクライブメソッドのリクエストに対する応答にはボディはないが，メッセージの最後にブランクラインが必要）

ヘッダ	
DATE	推奨項目．レスポンスメッセージが生成された日付．RFC 1123の日付
SERVER	必須項目．OS名，OSバージョン，UPnP/1.0，製品名および製品バージョンを連記したもの．文字列
SID	GENAが定義する必須ヘッダ．サブスクライブの識別子(ID)．必ずuuid:で始まる．UPnPベンダにより定義．単一のURI
TIMEOUT	必須項目．サブスクライブが期限切れになるまでの実際の期間．秒数または無限．UPnP Forum WCにより推奨される．UPnPベンダにより定義．1800秒(30分)より長く設定することが好ましいとされる．キーワードSecond-の後に整数値が続く(スペースなし)，あるいはinfinite(無限)

とが確認されます．

では，タイムアウトしてしまったら，どうなるのでしょうか？発行者はイベントメッセージの送信をやめるのと同時に，SID(サブスクライブ識別子)をリストから削除します．また反対に購読者は購読した発行者からのディスカバリメッセージを監視します．もし，ディスカバリメッセージ(以降DevMsgと記述)を受け取った場合，デバイスが購読をキャンセルしている可能性が大きいでしょう．その場合，再度購読する必要があります．

● サブスクライブメッセージ

デバイスディスクリプション(以降DevDescと記述)のサービスエレメント内にサービスのイベントURLがあります．そこへ購読者はサブスクライブメッセージを送ります．このメッセージの形式を解説します．まず，購読者が最初に送信するサブスクライブメッセージを**表10**に示します．

▶サブスクライブメッセージレスポンス

発行者が購読を許可するときは，購読者にSIDと購読のタイムアウト時間を割り当て，予定転送時間を含め30秒以内に**表11**の形式のレスポンスメッセージを送信します．次に発行者はイニシャルイベントメッセージを購読者へ送信します．このレスポンスメッセージを**表12**に示します．発行者が購読を許可できない場合，あるいはサブスクライブリクエストにエラーが発生した場合，発行者は**表11**のエラーのうちのいずれかを応答として送信します．レスポンスは，予定転送時間を含め30秒以内に送信されなくてはなりません．

▶更新サブスクライブメッセージ

以前リクエストした購読を更新するため，イベントURLへ送信します．サブスクライブメッセージと異なり，イベントメッセージを受信するためのURLを含みませんが，代わりにSIDが入ります．発行者はSIDで判断ができます．このメッセージを**表13**に示します．

発行者が購読の更新を許可するときは，サブスクライブメッ

〔表13〕更新サブスクライブメッセージ

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host: publisher port
<SPAN class=gena>SID: uuid: subscription UUID
T IMEOUT: Second-requested subscription duration
<BR>
```

（サブスクライブメソッドのリクエストにはボディはないが，メッセージの最後にブランクラインが必要）

リクエストライン		
SUBSCRIBE	GENAにより定義されたメソッド	
	publisher path	イベンティングURL(デバイスディスクリプションのサービスエレメントのeventSubURLサブエレメント)のパスコンポーネント．単一の相対URL
HTTP/1.1	HTTPバージョン	

ヘッダ	
HOST	必須項目．イベントURL(デバイスディスクリプションのサービスエレメントのeventSubURLサブエレメント)のドメイン名またはIPアドレスおよび任意のポートコンポーネント．ポートが空の場合，あるいは明記されていない場合，80が用いられる
SID	GENAが定義する必須ヘッダ．サブスクライブのID．サブスクライブリクエストへの応答としてパブリッシャーから割り当てられたSIDでなくてはならない．必ずuuid:で始まる．UPnPベンダにより定義．単一のURI
TIMEOUT	推奨項目．サブスクライブが期限切れになるまでの期間．秒数または無限．UPnP Forum WCにより推奨される．UPnPベンダにより定義．キーワードSecond-の後に整数値が続く(スペースなし)，あるいはinfinite(無限)

〔表14〕更新サブスクライブメッセージで発生するエラー

Incompatible headers
400 Bad Request．SIDヘッダおよびNTまたはCALLBACKヘッダのいずれかが存在する場合，発行者はHTTPエラー400 Bad Requestで応答する
Invalid SID
412 Precondition Failed．SIDヘッダの期限が切れていない，予定外の既知の購読に対応していない場合，発行者はHTTPエラー412 Precondition Failedで応答する
Missing SID
412 Precondition Failed．SIDヘッダが存在しない場合あるいは空の場合，発行者はHTTPエラー412 Precondition Failedで応答する
Unable to accept renewal
5xx．発行者が更新を許可できない場合，HTTP 500-シリーズのエラーコードで応答する

〔表16〕アンサブスクライブレスポンスで発生するエラー

Incompatible headers
400 Bad Request．SIDヘッダおよびNTまたはCALLBACKヘッダのいずれかが存在する場合，発行者はHTTPエラー400 Bad Requestでレスポンスする
Invalid SID
412 Precondition Failed．SIDヘッダの期限が切れていない既知の購読に対応していない場合，発行者はHTTPエラー412 Precondition Failedでレスポンスする
Missing SID
412 Precondition Failed．SIDヘッダが存在しない場合あるいは空の場合，発行者はHTTPエラー412 Precondition Failedでレスポンスする

〔表15〕アンサブスクライブメッセージ

UNSUBSCRIBE publisher path HTTP/1.1 HOST: publisher host: publisher port <SPAN class=gena>SID: uuid: subscription UUID （アンサブスクライブメソッドのリクエストにはボディはないが，メッセージの最後にブランクラインが必要）		
リクエストライン		
UNSUBSCRIBE	GENAにより定義されたメソッド．サブスクライブをキャンセルする	
	publisher path	イベンティングURL（デバイスディスクリプションのサービスエレメントのeventSubURLサブエレメント）のパスコンポーネント．単一の相対URL
HTTP/1.1	HTTPバージョン	
ヘッダ		
HOST	必須項目．イベンティングURL（デバイス記述のサービスエレメントのeventSubURLサブエレメント）のドメイン名またはIPアドレスおよび任意のポートコンポーネント．ポートが空の場合，あるいは明記されていない場合，80が用いられる	
SID	GENAが定義する必須ヘッダ．サブスクライブのID（識別子）．サブスクライブリクエストへのレスポンスとしてパブリッシャーから割り当てられたサブスクライブIDでなくてはならない．必ずuuid:で始まる．UPnPベンダにより定義．単一のURI	

セージのレスポンスと同じ形式のメッセージを30秒以内（予定転送時間を含む）にレスポンスメッセージを送信します．発行者が購読の更新を許可できない場合，あるいはサブスクライブリクエストにエラーが発生した場合，発行者は**表14**のエラーのうちのいずれかを応答として送信します．レスポンスは，予定転送時間を含め，30秒以内に送信されなくてはなりません．

▶アンサブスクライブ

購読者は，発行者の状態をチェックすることが不要になると，購読をキャンセルします．このメッセージの形式を**表15**に示します．

▶アンサブスクライブレスポンス

購読をキャンセルするために，発行者は予定転送時間を含め30秒以内に以下の形式でレスポンスしなければなりません．

```
HTTP/1.1 200 OK
```

（NOTIFYメソッドのリクエストにはボディはないが，メッセージの最後にブランクラインが必要）

キャンセルリクエストにエラーが発生した場合，発行者は**表16**のエラーのうちのいずれかをレスポンスします．

● イベントメッセージ

ここでイベントメッセージの形式を説明しますが，その前にevent keyに関してだけ説明します．発行者は，イベントメッセージを購読者へ送るたびにevent keyをインクリメントします（イニシャルイベントメッセージを送付したときに0にリセットされる）．これを購読者はチェックします．たとえば，event keyが3のイベントメッセージの後に6のメッセージを購読者が受け取れば，4と5のイベントメッセージを受信し損ねています．このようなとき，購読者はその購読をキャンセルし，再度購読する必要があります．イベントメッセージの形式を**表17**に示します．

▶イベントメッセージレスポンス

購読者がイベントメッセージを受信したことを発行者へ知らせるため，予定転送時間を含め30秒以内に以下の形式のレスポンスメッセージを送信します．

```
HTTP/1.1 200 OK
```

（NOTIFYメソッドのリクエストにはボディはないが，メッセージの最後にブランクラインが必要）．

イベントメッセージにエラーが発生した場合，購読者は**表18**のエラーのうちのいずれかをレスポンスとして送信します．レスポンスは，予定転送時間を含め30秒以内に送信されなくてはなりません．

プレゼンテーション

プレゼンテーションは，UPnPネットワークにおけるステップ5です．コントロールポイントがディスカバリ（ステップ1），デバイスとサービスのディスクリプション（ステップ2）を終了した時点で，コントロールポイントはプレゼンテーションの

〔表17〕イベントメッセージ

```
NOTIFY delivery path HTTP/1.1
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml
CONTENT-LENGTH: Bytes in body
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: event key

<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
 Other variable names and values(if any)go here.
</e:propertyset>
```

項目			説明
リクエストライン			
NOTIFY			GENAにより定義されたメソッド．クライアントにイベントについて告知する
	delivery path		配信URL（サブスクライブメッセージのCALLBACKヘッダ）のパスコンポーネント．イベントメッセージの送信先．単一の相対URL
HTTP/1.1			HTTPバージョン
ヘッダ			
HOST			必須項目．配信URL（サブスクライブメッセージのCALLBACKヘッダ）のドメイン名またはIPアドレスおよび任意のポートコンポーネント．ポートが空の場合，あるいは明記されていない場合，80が用いられる
CONTENT-LENGTH			必須項目．バイト単位で示したボディの長さ．整数
CONTENT-TYPE			必須項目．text/xmlでなくてはならない
NT			GENAが定義する必須ヘッダ．通知タイプ（Notification Type）のこと．UPnP:eventでなくてはならない
NTS			GENAが定義する必須ヘッダ．通知サブタイプ（Notification Sub Type）のこと．UPnP:propchangeでなくてはならない
SID			GENAが定義する必須ヘッダ．サブスクライブのID（識別子）．必ずuuid:で始まる．UPnPベンダにより定義．単一のURI
SEQ			UPnPが定義する必須ヘッダ．イベントキー．イニシャルイベントメッセージの場合キーは0となる．特定のサブスクライブバーに送信される各イベントメッセージに対し1ずつインクリメントする．長さ8バイト．オーバフローを防ぐため一つのインテジャー型になる．単一の整数
ボディ			
propertyset			必須項目．xmlnsネームスペース属性は，必ずurn: schemas-UPnP-org:event-1-0になる．すべてのサブエレメントはこのネームスペースにより認可されたもの．以下のサブエレメントを含む
	property		必須項目．イベントメッセージの各変数名および値に対し1度繰り返す
	propertyset		ネームスペースにより認可されたもの．以下のサブエレメントを含む
		variableName	必須項目．エレメントは，変化した状態変数の名前（サービスディスクリプションのstateVariableエレメントのnameサブエレメント）．propertysetネームスペースにより認可されたもの．Valuesは，この状態変数の新しい値．UPnPサービスディスクリプションで指定される単一のデータ型

処理に移ることができます．コントロール（ステップ3）イベンティング（ステップ4）と連携しますが，実行されている必要はありません．プレゼンテーションは，どちらかというとコントロールやイベントの補助的なものです．プレゼンテーションによって，デバイスがもつWebサービスをIEなどのWebブラウザで表示することができます．これによりWebページからのデバイス制御，サービスの状態変数を知ることができます．プレゼンテーションの動作イメージを**図5**に示します．

プレゼンテーションのためのURLは，DevDescのpresentationURLエレメントに含まれています．プレゼンテーションページを取得する場合，コントロールポイントはHTTP GETリクエストをプレゼンテーションURLに対して発行し，デバイスがプレゼンテーションページを返します．UPnP Device/Service Template，標準デバイスおよびサービスタイプと異なり，プレゼンテーションページの機能はUPnPベン

〔表18〕イベントメッセージレスポンス時に発生するエラー

エラー	説明
Missing SID	412 Precondition Failed．SIDヘッダが存在しない場合，あるいは空の場合，購読者はHTTPエラー412 Precondition Failedでレスポンスする
Invalid SID	412 Precondition Failed．SIDヘッダが期限の切れていない既知の購読に対応していない場合，購読者はHTTPエラー412 Precondition Failedでレスポンスする（サービスは，このエラーメッセージを受け取った場合このSIDを破棄する）
Missing NT or NTS header	400 Bad Request．NTまたはNTSヘッダが存在しない場合，購読者はHTTPエラー400 Bad Requestでレスポンスする
Invalid NT header	412 Precondition Failed．NTヘッダがUPnP:eventでない場合，購読者はHTTPエラー412 Precondition Failedでレスポンスする
Invalid NTS header	412 Precondition Failed．NTヘッダがUPnP:prochangeでない場合，購読者はHTTPエラー412 Precondition Failedでレスポンスする

〔図5〕プレゼンテーションの動作

ダにより自由に使用できます．プレゼンテーションページは，UPnP Forum WCの対象ではありません．Webページは，必ずバージョン3.0以降のHTMLページでなくてはなりません．しかし，他のデザイン要因はベンダが指定できます．これにはコントロールポイントのWebブラウザの全機能，使用するスクリプト言語またはブラウザプラグイン，そしてデバイスとの対話手段などが含まれます．

プレゼンテーションページを配置するため，UPnPベンダはコントロールやイベンティング，デバイスの既存機能の強化などのためのUPnPメカニズムを用いることができますが，それ以外の選択肢もあります(よく聞く質問で，プレゼンテーションで実行される制御はUPnPのコントロールなどを使用するべきか？というものがあるが，答えはNoである．独自でもコントロールを使ってもかまわない)．

プレゼンテーションページでは，HTMLが提供するローカライズのためのメカニズム(例：charset属性をもつMETAタグなど)を用いる必要があります．コントロールポイントでは，ローカライズされたプレゼンテーションページを取得するためにHTTPのACCEPT-/CONTENT-LANGUAGE機能を用います．具体的には，コントロールポイントはプレゼンテーションページに対するリクエストにHTTPのACCEPT-LANGUAGEヘッダを含めることができます．ACCEPT-LANGUAGEヘッダがリクエストに含まれている場合，レスポンスには必ずページの言語を識別するためのCONTENT-LANGUAGEヘッダが含まれます．

まとめ

Universal Plug and Play Device Architecture V1.0のスペックの話をしてきましたが，詳細に関してはスペックそのものを参照してください．これまでの話は，UPnPデバイスアーキテクチャV1.0や各DCPのスペックを理解する上での助けになると思います．なお，機会があれば，各DCPのスペックの解説やWindows XPのUPnP APIの紹介をしたいと考えています．

参考文献

1) Universal Plug and Play Device Architecture V1.0, http://www.upnp.org/download/UPnPDA10_20000613.htm
2) Simple Service Discovery Protocol(SSDP), http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt
3) Multicast and Unicast UDP HTTP Messages, http://www.upnp.org/download/draft-goland-http-udp-04.txt
4) General Event Notification Architecture(GENA), http://www.upnp.org/download/draft-cohen-gena-client-01.txt
5) Sample Device and Service Templates, http://www.upnp.org/resources/devices.asp
6) Windows XP の Universal Plug and Play, http://www.microsoft.com/japan/windowsxp/pro/techinfo/planning/upnp/UniversalPlugandPlayinWindowsXP.doc
7) Simple Object Access Protocol(SOAP), http://www.w3.org/TR/SOAP/, http://www.microsoft.com/japan/developer/workshop/xml/general/soapspec.asp
8) Extensible Markup Language(XML), http://www.w3.org/XML/

ちゃま・やすし テクニカルライター

PC/ATのさまざまな資源を管理する

ACPIによるPC/ATの電力管理とコンフィグレーション

安達健一

Advanced Configuration and Power Interface

従来，PC/ATの電力管理にはAPM（Advanced Power Management）が使われてきた．しかし，昨今のハードウェアの進化によって，APMでは要求に応じられないような局面も出てきた．
そこで登場したACPI（Advanced Configuration and Power Interface）は，電力管理はもちろんのこと，多様な資源のコンフィグレーションを可能にしている．
本稿では，注目が集まっているACPIに関して，実例をまじえて解説を行う．（編集部）

最近のPCでは，パワーマネジメントとリソースコンフィグレーションを実現する土台として，ACPIが必要不可欠となってきています．本稿では，ACPIの核となる基本コンセプトを平易に解説し，難解とされるACPI仕様書を読み解く道しるべを示します．

ACPIの基本コンセプト

現時点では，ACPIに未対応のPCもありますが，最近の機種であれば原則としてACPIに準拠しています．さらに，一部にはパワーマネジメントとリソースコンフィグレーションの基盤として，ACPIしかサポートしないモデルも登場し始めています．

そこで，ここではなぜACPIが必要とされるのか，ACPIの基本コンセプト/ACPIへといたる歴史を簡単に確認しておきます．

ACPIという言葉は，Advanced Configuration and Power Interfaceの略で，その名が示すようにConfiguration：リソースコンフィグレーションと，Power：パワーマネジメントを司るうえで，プラットホームハードウェアとシステムソフトウェアのインターフェースとなるものです．

● ACPI以前 ── BIOSへの依存

ACPIが登場する以前から，これらの機能はPnP BIOSやAPM BIOSの規格にのっとり，OSとBIOSが協調する形で実現されていました．

しかし，その方法はBIOSが「主人公」に近い位置を占め，システムボードに関わる情報の収集・管理から意思決定，そして具体的な処理の実行にいたるまで，すべてがBIOSに深く依存したものでした．

したがって，これらACPI以前の方式でリソースコンフィグレーションやパワーマネジメントを行う場合，OS上で実行される通常のプロセスから見て問題となる点がいくつかありました．

● システム管理モード

なかでも，OSの実行中にランタイムでシステム管理割り込み（SMI：System Management Interrupt）を発行し，システム管理モード（SMM：System Management Mode）という，OSから「見えない」モードでBIOS側の用意したハンドラを実行するしくみは，危険性が高いものです．

なぜなら，システム管理モードではOSのすべての処理の実行は，ごくわずかな時間ですが停止させられてしまい，このとき実行されるBIOS側のコードに障害があればシステムのハングアップといった致命的な問題が発生し，OS側では対処する術がないからです（**図1**）．

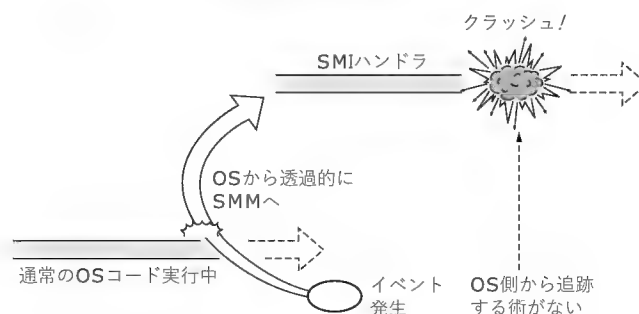
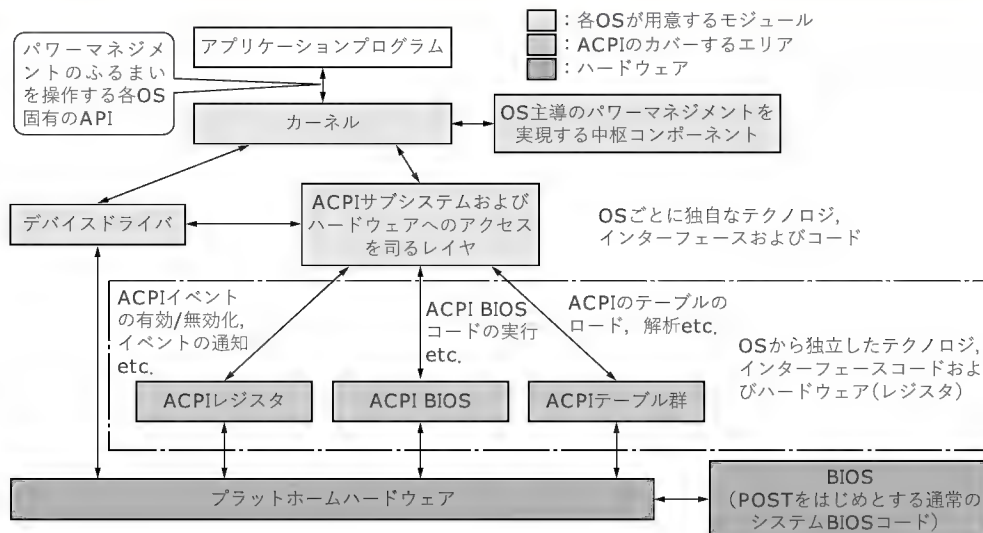
また，システム管理モードはOSからは透過的であり，本来であればOSの実行環境に影響を与えないはずですが，濫用されるとOS上のコンテクストの整合性が損なわれてしまうこともありました．

● ACPI ── OSによる中央集権的な管理

そこで，ACPIではOSPM ── Operating System Directed Configuration&Power Managementという，リソースコンフィグレーションやパワーマネジメントに関わるすべての情報をOSに集約し，BIOSは具体的な手段を提供するだけで，その手段の実行はOSの意思決定のもと，OSの制御下で行われるというコンセプトが取り入れられました（**図2**）．

ACPIでは，BIOSがASL（ACPI Source Language）という言語を用いて，ACPI管理下にあるデバイス間の論理関係や制御手順を記述し，BIOS ROMに格納しておきます．

そして，システム起動時にBIOSがこれらのACPI関連情報をメインメモリに展開し，OSが立ち上がるとOS内に実装されたACPIサブシステムがBIOSから渡された情報を解析して，パワーマネジメントとリソースコンフィグレーションを実現する基盤となる情報を得ます．

〔図1〕SMIハンドラで重大な障害が発生

〔図2〕ACPIの基本コンセプト

これ以降は，OSが中央集権的にパワーマネジメントやリソースコンフィグレーションに関する情報の収集/管理から意思決定，そして具体的な処理の実行にいたるまでをコントロールします．

ACPI対応システムでは，OS実行中にプラットホームハードウェア上で，たとえば電源ボタンが押されたり，温度があるしきい値を超えたり，バッテリが取り外されたり……といったACPIの管理対象であるイベントが発生すると，ACPIのコアロジック部分(通常はチップセット内のレジスタとして実装)を経由してSCI：System Control Interruptという割り込みがかかります．

OSはSCIを受け付けると，ACPIイベントのソースを特定し，必要に応じてBIOSが提供するコントロールメソッドと呼ばれるコードを実行するなどしてACPIイベントを適切にハンドルします．

このACPIイベントのハンドリングが，ACPIの基本コンセプトに基づき，OSの制御下で，OSから「見える」モードで実行される点がポイントです．

ACPI対応OSの現状

ハードウェア/BIOSについては前述のとおり，現在ほとんどのPCがACPIに準拠しています．一方，ACPI対応OSとしては，マイクロソフトのWindows 2000以降のNTカーネルベースのOSがACPI仕様書1.0bに準じた完成度の高いACPIサポートを実現しています．

同社の最新クライアントOSであるWindows XPでは，プロセッサドライバなどACPI2.0のいくつかの新機能も追加サポート[注1]され，ACPIを全面的に利用したシステムワイドなパワー

コラム1

ACPIへの道程

今から10年ほど前，インテルがPCにはじめてパワーマネジメントという概念を取り入れようとしたとき，業界からはほとんど相手にされなかったと聞きます．

しかし，その後，当時“Wintel”ともいわれ，蜜月関係にあったマイクロソフトの協力をとりつけたことで，PCベンダも理解を示し始め，1990年代後半にはAPM規格がノート型コンピュータでは積極的に採用されるようになりました．

しかし，APMは本稿で述べたように，BIOSが意思決定にまで介入するという性格であったこと，さらには機能拡大につれてベンダ間でのバラつきや，BIOS ROMの容量を圧迫するといった課題が顕在化してきました．

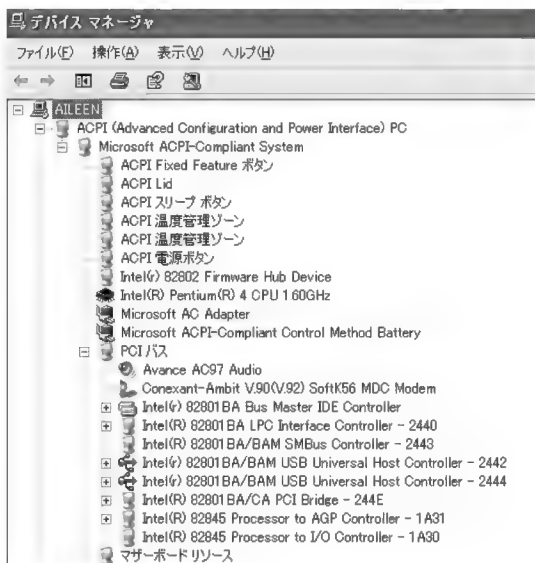
ACPIは，そうしたAPMの問題点をふまえ，システムの状態についてもっとも正確な情報を知り得るOSが，中央集権的にパワーマネジメントを司るアプローチを取っています．

ただし，ACPIはたんにAPMを置き換えるだけのものではなく，APMと同様に，限界が見え始めていたPnP BIOSや，本稿ではふれていませんが，マルチプロセッサシステムで採用されていたMPSなどの規格についても，BIOS主導からOS中心に切り替える役割を果たすものです．

ACPI仕様書を細かく見ていくと，じつに多彩な内容がカバーされた規格であることがわかります．見落とされがちですが，IDE，FDDなどのニッチな部分についても標準化を試みていたりと，とても意欲的です．そして，そのどれにも共通しているのが，BIOS依存部分をOS主導に，というパラダイムの転換です．

注1：ACPIテーブルの64ビットフィールド対応，プロセッサ省電力テクノロジ，PCIホットプラグの3機能だけが新たに実装されている．

〔図3〕Windows XP デバイスマネージャの画面

〔図4〕ACPIサブシステムの位置づけ

マネジメントとリソースコンフィグレーションを達成しています(**図3**).

同じマイクロソフトのOSでも，コンシューマ向けのWindows 98 SEやWindows Meでは，当初の開発時点でハードウェア/BIOS側のACPIサポートがまだ未成熟であったこともあり，一部ACPI仕様から外れた実装方式[注2]でACPIサポートが実現されています.

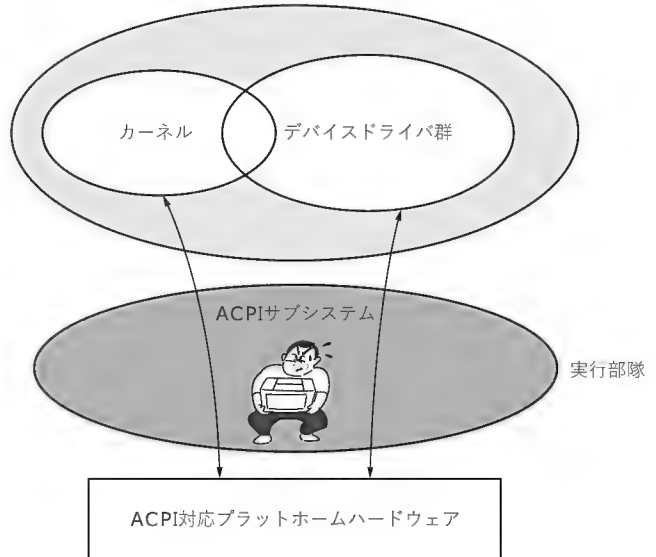
また，PC UNIXの中にも，積極的にACPIサポートを実現しようとする活動が広がっています．たとえばLinuxでは，カーネル2.5.xからACPIサブシステムのコードがツリーに標準で入り，インテルのエンジニアがリーダーシップを取ってACPIサポートを完成させつつあります.

おもだったLinuxディストリビュータの中にも，Red Hatなどのように自社のパッケージでACPIの標準サポートを表明するところが出てきました.

● ACPIサブシステムの位置付け

OSがより高度なパワーマネジメントとリソースコンフィグレーションを実現するとき，まずはカーネルやデバイスドライバ側にこうした機能を意識したインフラストラクチャが確立されていることがたいへん重要となります.

ACPIサブシステムは，実行部隊としてACPIイベントを検出したり，カーネルやデバイスドライバの判断・意思決定の結果として実行が必要となったタスク，たとえばシステムやデバイスをスリープステートに落とす，あるいはACPI管理下のデバイスを列挙して有効にし，リソースを割り当てるといった作業を遂行するためにハードウェア資源にアクセスするレイヤです(**図4**).

注2：一部のACPIメソッドは使用を推奨しないなどといった措置がとられた.

● Windows Driver ModelとACPI

Windows XPをはじめとするマイクロソフトの最近のOSでは，カーネルがパワーマネジメントやリソースコンフィグレーションについて中央集権的に支配しています．また，デバイスドライバもWindows Driver Model(WDM)に即して，パワーマネジメントおよびリソースコンフィグレーションを統一された方式でサポートしています.

このため，システムスタンバイや休止状態という，とくにノートPCユーザーから強い要望のある機能についても，カーネルと各デバイスドライバがパワーマネジメント用のIRP(I/Oリクエストパケット)を介し一貫性のある処理を進め，必要な部分でACPIサブシステムを利用して目的を達成します(**図5**).

カーネル側の意思決定を担う部分とACPIサブシステムとはきちんと分離されていて，カーネル側はACPIを利用して情報を収集して判断を下し，ACPIサブシステムは実行部隊としての役割に徹することができます.

● Intel ACPI Component Architecture

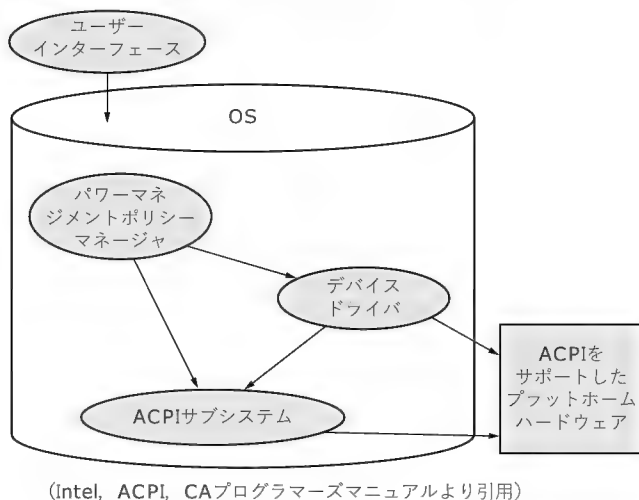
インテルが，現在Linux上で開発に注力しているACPIサブシステム(ACPI Component Architecture)もカーネル部分から独立した実行部隊として設計され，さらに理論上はホストOS非依存の実装を目指しています.

しかしながら，WindowsのようなプロプライエタリなOSと違ってオープンソース系のOSでは，カーネル中枢に新たな機能を追加したり，デバイスドライバにパワーマネジメントやリソースコンフィグレーションを意識した統一の約束事を徹底することが容易ではありません.

とりわけLinuxのような，ポピュラーで関係する開発者の多

〔図5〕Windows Driver ModelとACPI

〔図6〕LinuxにおけるACPIコンポーネントアーキテクチャ

(Intel, ACPI, CAプログラマーズマニュアルより引用)

いプロジェクトでは，この傾向が強くなります．カーネル2.5以降の試みとして，Linux Driver ModelというWindowsのWDMに似たデバイスドライバ実装の指針が検討されているようですが，少なくとも現時点では強固なインフラストラクチャは確立されていません(**図6**)．そのため，WindowsのようにACPIをきちんと利用したシステムスタンバイや休止状態を実現するにはまだ時間を要しそうです．

ACPI仕様書

冒頭に述べたように，「ACPI仕様書は難しい」という声をよく耳にします．その理由として，以下の点を指摘できます．

① 構成上の問題

扱う対象の種類が多く，範囲も広いため，関連するさまざまな話題を寄せ集めたつくりとなっていて，全体を見渡すのが難しい．

② 内容の一貫性

総論と各論が混在していて，たとえばUSB仕様書などのように一つの規格に特化したものと比べて読み通しづらい．

③ 背景知識の必要性

ハードウェアやBIOSなどの前提知識については解説していないことが多い．また，PnP BIOS，APM BIOSなど，過去の類似規格からの歴史的な経緯や，PCIを筆頭に関連の深いバス規格を知らないと理解するのが困難．

そこで，ここではACPIというメカニズムの骨格となる重要な部分，言い換えるとパワーマネジメントとリソースコンフィグレーションの双方をACPIの方式で実現するためのベースになるしくみに注目し，必要な情報を補足しながら解説します．

これにより，ACPIという世界に関して全体の見通しを良くしたいと思います．ACPI仕様書内でいうと，主として"4 ACPI HARDWARE SPECIFICATION"，"5 ACPI SOFTWARE PROGRAMMING MODEL(**表1**)"に対応しますが，必ずしもその記述内容を忠実に一つ一つ取り上げるわけではありません．

説明する順序としては，おおむね次のように考えています．

① ACPIテーブル

ACPI対応BIOSがシステムボードに関する情報を収集・整理してOSに渡すACPIテーブル群の規定．

② ACPIの初期化

OSがACPIモードを有効にし，BIOSから渡された情報を用いてACPIネームスペースを展開していく処理．

③ ACPIイベント

ACPIイベントをOSに通知するためのハードウェア的なしかけ(チップセットに実装されたレジスタ群など)とそのハンド

〔表1〕ACPI仕様書の構成

1 INTRODUCTION	ACPI規格のめざすゴールの提示
2 DEFINITION OF TERMS	用語の定義
3 OVERVIEW	ACPIを利用する典型的な事例の紹介
4 ACPI HARDWARE SPECIFICATION	ACPIを実現するためのハードウェア上のしかけ
5 ACPI SOFTWARE PROGRAMMING MODEL	ACPIのゴールを達成するためのソフトウェア側のしくみ
6 CONFIGURATION	デバイスの列挙やリソースコンフィグレーションについての概要
7 POWER AND PERFORMANCE MANAGEMENT	個々のデバイスおよびシステム全体のパワーマネジメントについての概要
8 PROCESSOR CONTROL	プロセッサの省電力テクノロジの標準化
9 WAKING AND SLEEPING	システムスリープ/ウェイクの具体的な処理フロー
10 ACPI-SPECIFIC DEVICE OBJECTS	ACPIが特別にケアするニッチなデバイスの取り扱いその他
11 POWER SOURCE DEVICES	電力の供給源すなわちバッテリおよびACアダプタについて
12 THERMAL MANAGEMENT	ファンおよびCPUスロットリングによるACPIでの放熱管理
13 ACPI EMBEDDED CONTROLLER INTERFACE SPECIFICATION	エンベデッドコントローラの役割とACPIでの通信方法
14 ACPI SYSTEM MANAGEMENT BUS INTERFACE SPECIFICATION	SMBusの役割とACPIでの通信方法
15 SYSTEM ADDRESS MAP INTERFACES	システムメモリのアドレスマップ取得方法など
16 ACPI SOURCE LANGUAGE (ASL) EFERENCE	ASL言語リファレンス
17 ACPI MACHINE LANGUAGE (AML) SPECIFICATION	AMLコードの厳密な定義
APPENDIX A : DEVICE CLASS PM SPECIFICATIONS	デバイス種別ごとのデバイス電力ステートに関する定義
APPENDIX B : ACPI EXTENSIONS FOR DISPLAY ADAPTERS	ACPIによるビデオカードの出力先制御や輝度調整などの補完

リング.

こうした基本事項を身に付けたあと，ACPIの管理下で達成される具体的な動作の中で，もっとも代表的な「スタンバイ」，「休止状態」(**図7**)など，システム全体のスリープ/ウェイクについて解説します.

また，ACPI 2.0で新たに標準化されたプロセッサのパフォーマンス制御についても取り上げる予定です.

ACPI 2.0では，インテルの最近のモバイル系プロセッサにおけるSpeedStepやAMD社Athlon/DuronでのPowerNow!，Transmeta社のCrusoeでのLongRunといったプロセッサ省電力テクノロジが新たに標準規格化され，ACPIのコンセプトにしたがいOSによるネイティブ制御の対象となりました.

● ACPI仕様書の入手方法

ACPI仕様書は，ACPIオフィシャルサイト[1]から入手が可能です．現在公開されている最新版はバージョン2.0bですが，これは2.0のマイナアップデートなので，本稿では基本的に2.0を基準にします.

標準規格の仕様書という性格から，ACPI仕様書の記述は一般的すぎて，具体的にイメージすることが難しい場合があります．そのようなとき，ACPI準拠のチップセットのデータシートをチップセットベンダのWebサイトから入手し，ACPI仕様書での規定が実際のハードウェアでどのように実装されているのかを確認してみるとよいと思います.

また，ACPI仕様書での用語の定義は，現実世界のハードウェアの実装の変化にともない，実態にあわせて少し読み換えるなど柔軟に解釈する必要があります.

なお，はっきりとアナウンスはされていませんが，今後ACPI“3.0”が策定されていくときには，とくにハイエンドのサーバ機における高度なリソースコンフィグレーションを意識した内容や，ACPI 2.0の段階ではまだ市場に出回っていなかったテクノロジで関連の深いもの(ハイパースレッディングやグラフィクスコントローラの省電力機能など)がACPI規格に取り込まれていくのではないかと思われます.

〔図7〕Windows XPでシステムスリープへの移行を選択

ACPIテーブル

ACPI BIOSは，OSがACPI対応PCを管理するために必要な情報をACPIで規定されたテーブルの形式で提供します．テーブルには，プラットホームハードウェアに備わるACPIレジスタなど基本的な能力について定義したものや，システムボード上のデバイスの論理関係(ACPIネームスペース)を表現し，個々のデバイスをコントロールする手順(コントロールメソッド)を記述したものなど各種存在します(**表2**).

次に，そのうちのキーになるテーブルについて解説を加えて

〔表2〕ACPIの主要なテーブル

テーブル名	シグネチャ	詳細名称	役　割
RSDT	"RSDT"	Root System Description Table	他のテーブルの格納場所をOSに知らせる
FADT	"FACP"	Fixed ACPI Description Table	ACPI関連の基本的な能力を定義
FACS	"FACS"	Firmware ACPI Control Structure	OSとBIOSのハンドシェイクをとりもつストラクチャ
DSDT	"DSDT"	Differentiated System Description Table	ACPIネームスペースの基盤
……	……	……	……
MADT	"APIC"	Multiple APIC Description Table	マルチプロセッサ構成での割り込みコントローラ(APIC)に関する設定その他
ECDT	"ECDT"	Embedded Controller Boot Resources Table	起動時の早い時点でエンベデッドコントローラにアクセスする場合に必要

いきます.

● RSDP StructureとRSDT

ACPIテーブルのうちRSDT(Root System Description Table)は，ほかのテーブルの格納場所をOSに知らせる案内板の役割をもちます.

IA-32の場合，OSは起動時にメインメモリのBIOS領域内でRSDP(Root System Description Pointer) Structureをそのシグネチャである"RSD PTR"でサーチし，見つかったRSDP Structure内からRSDTの物理アドレスを得ます(**リスト1**).

こうしてアクセスが可能となったRSDTの末尾にあるEntry

コラム2

PC UNIXでのACPIへの対応

Linux上でのACPIサポートは，現在SourceForgeの"ACPI4Linux"プロジェクト(http://acpi.sf.net/)として，アクティブに開発されています．カーネル2.5.xであれば，ACPIのコードは標準でツリー(drivers/acpi)に含まれていますが，最新版でテストしたいときは，同サイトからパッチをダウンロードして適用し，任意にコンフィグレーションしてカーネルを再構築します.

うまく再構築ができて，ACPIサポート状態でブートできると，/procファイルシステムにACPI関連のエントリが出現します．ACPI経由でバッテリなどの情報を得たり，ACPIイベントの監視や，スリープステートあるいはプロセッサのパフォーマンスステートをコントロールするには，/proc/acpi配下のエントリを操作します.

/proc/acpiインターフェースの詳細については，上述の"ACPI4Linux"SourceForgeサイトのDocumentationを参考にしてください．今のところ，一般のユーザーが/procファイルシステムを意識せずに済むようなユーザーランドのツールは，ごくシンプルなものしか出回っていません.

またLinuxの場合，本来のACPIのコンセプトが，現時点ではクリーンに実現されていないため，一般にパワーマネジメント機能として期待されることを行うには，別途，SwSusp：Software Suspendや，CPUFreq：CPU Frequency Driverといった別プロジェクトの成果物を必要とすることがあります.

残念ながら，こうした各プロジェクトの成熟度やインテグリティは，現時点では，とりわけノート型コンピュータのユーザーにとっては満足のいくものではありません．しかし，最近，IntelとRed Hatの間でもめていたAMLインタプリタ部分についてのライセンス問題がデュアルライセンスという形で決着し，今後，Red Hatをはじめメジャーなディストリビュータが ACPIを標準サポートしていくことで，開発がよりいっそう促進されるものと思われます.

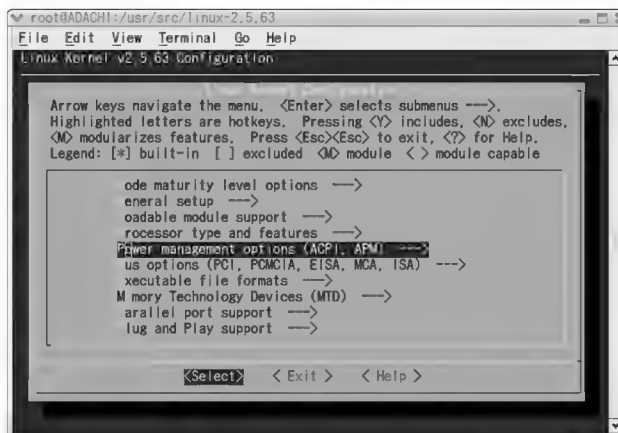
●Intel/Red Hat，専有技術とオープンソースの融合でライセンス問題を解決
http://www.zdnet.co.jp/news/0302/18/nebt_01.html

なお，FreeBSDとNetBSDについて，リサーチした範囲でごく控えめにコメントしますと，どうやら両者とも，本文で言及しているIntel ACPI CAサブシステムをうまく活用し，カーネルやデバイスドライバ部分は独自に発展させて，ACPIサポートを実現しているようです.

とくにFreeBSDは，日本の開発者が牽引して，Linuxを上回るクオリティを達成していると聞きました．BSD系のPC UNIXも試してみたいとは思いつつ，今のところ利用経験がないため，今回はこれ以上のコメントは差し控えておきます;-)

参考文献

1) ACPI-JP for FreeBSDのメーリングリスト
http://www.jp.freebsd.org/acpi/ml-j.html

〔図A〕Linuxカーネル再構築のようす

フィールドは，他のテーブルへのポインタの配列となっており，RSDTから各テーブルがたどれるようになります(**図8**)．

なお，これらのACPIテーブルは，通常，物理メモリの最上位の部分に展開されます．

● FADT

RSDTがポイントするテーブルのうち，要となるテーブルがFADT(Fixed ACPI Description Table)で，プラットホームハードウェアが有するACPI関連の基本的な能力が定義されています．

そこには，ACPIモードを有効にするために必要な情報や，ACPIイベントをOSに通知するSCI —— System Control Interruptという割り込みを入れるACPIコアロジックのハードウェア上の実装に関する情報などが含まれています(**表3**)．

そのほか，FADTは，FACS(Firmware ACPI Control Structure)，DSDT(Differentiated System Description Table)という二つの重要なアイテムへのポインタも含んでいます．

● FACS

FACSは，OSとBIOSコードのハンドシェイクをとりもつ情報を含んでいます(**表4**)．

Waking Vectorにはシステムウェイク時，OSに制御が戻った直後，リアルモードで実行されるコードが格納された物理アドレスがセットされます．

たとえば，RAMにコンテクストを保存してスリープ(Suspend-to-RAM：S3)したシステムに対し，電源ボタンを押し下げなどのスリープ解除のイベントを発生させると，まずハードウェアレベルでシステムがウェイクアップし，CPUリセット後，BIOSの簡易初期化コードが走り，次にWaking Vectorにジャンプして OSの用意したコンテクスト復元用のルーチンに制御が渡されます(**図9**)．

また，Global Lockは，OSのコードとSMIハンドラの双方によってアクセスされ得るハードウェア資源に対するアクセスの同期を取るために利用されます．

ACPIのコンセプトにおいてはSMIへの依存から脱却することが期待されますが，現実にはACPI対応システムであっても，OS実行中にSMIトラップが利用されることがあります．

● DSDT

システムボード上でACPIが管理するデバイスツリーおよび個々のデバイスやイベントのコントロール手順を表現したテーブルがDSDTです．

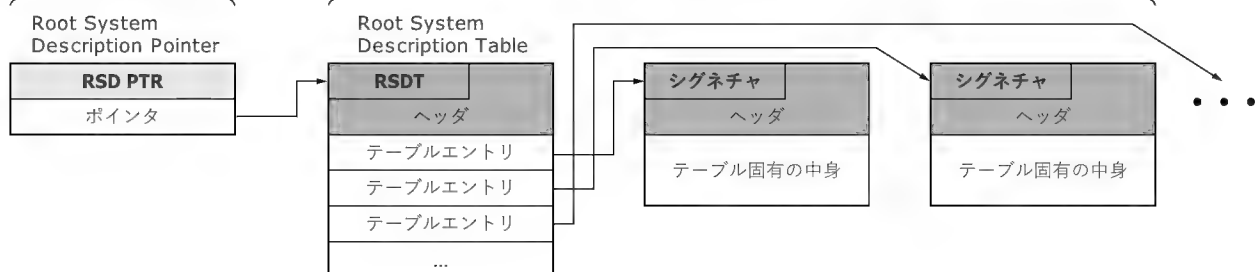
DSDTがロードされた後に，別途副次的なテーブルやデータブロックを追加することもできますが，DSDTは起動時にロードされたあとはアンロードされることがなく，DSDTがすなわちACPIネームスペース全体を指すこともあります．

ACPIでは，プラットホームハードウェアの実装についてASL(ACPI Source Language)という専用の言語を用いて記述します．そして，このASLをプラットホームに依存しない中間言語(バイトストリーム)に翻訳したものが，AML(ACPI Machine Language)です．DSDTは，AMLの形式でBIOS ROMに格納されています．

OSがランタイムでACPI関連処理を行う際には，ACPIサブ

〔リスト1〕DOS窓でDebugコマンドを使ってRSDPを見つけ，RSDTの格納位置を知る

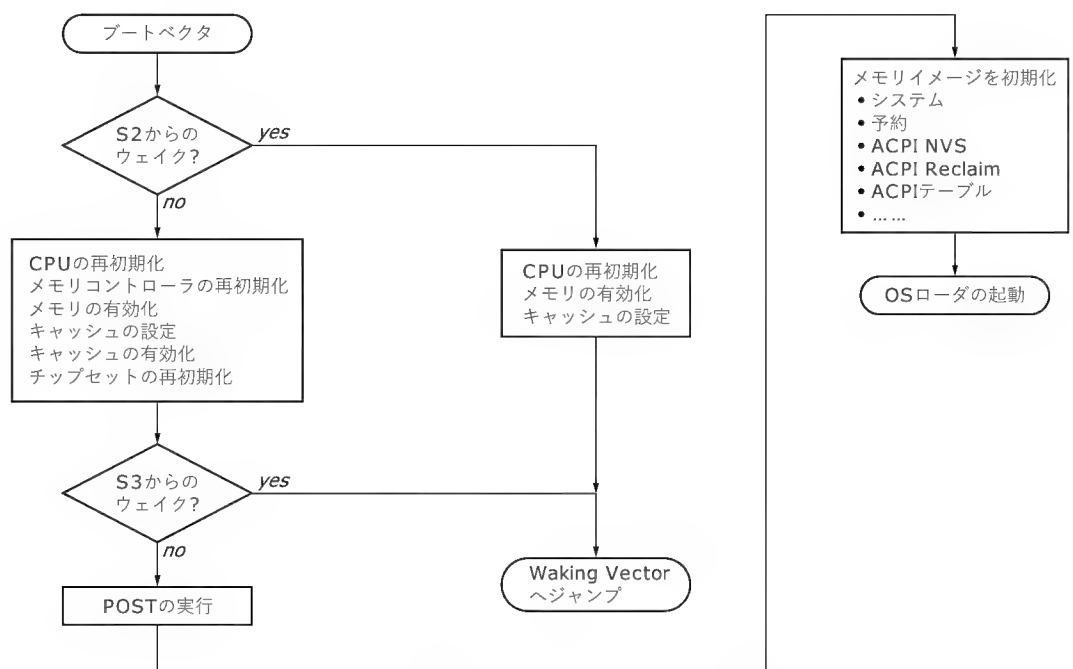
```
C:¥> DEBUG
-S E000:0 Lffff "RSD PTR "
-S F000:0 Lffff "RSD PTR "
F000:7120
-D F000:7120 L24
F000:7120  (1)52 53 44 20 50 54 52 20-9E (2)43 51 50 55 42 20 02   (1)RSD PTR .(2)CQPUB   .
F000:7130  (3)C0 33 F7 1F 24 00 00 00-F8 33 F7 1F 00 00 00 00   .3..$....3......
F000:7140  9B 00 00 00

(1) RSDP Structure のシグネチャ
(2) OEM ID
(3) RSDT の物理アドレス 0x1FF733C0
```

〔図8〕RSDP構造体とRSDTおよびその他のテーブル

〔表3〕FADTテーブルの主要フィールド

フィールド名	備　考
FIRMWARE_CTRL	FACSストラクチャの物理アドレス
DSDT	DSDTテーブルの物理アドレス
……	……
SCI_INT	ACPI SCI割り込みにアサインされたIRQ(8259互換PICカスケード接続の場合)
SMI_CMD	SMIコマンドポート(I/Oポート)のアドレス
ACPI_ENABLE	ACPIモードを有効にする値
……	……
PM1a_EVT_BLK	PM1a Event Register BlockのI/Oポートアドレス(固定イベント用)
PM1b_EVT_BLK	PM1b Event Register BlockのI/Oポートアドレス(固定イベント用)
PM1a_CNT_BLK	PM1a Control Register BlockのI/Oポートアドレス
PM1b_CNT_BLK	PM1b Control Register BlockのI/Oポートアドレス
PM2_CNT_BLK	PM2 Control Register BlockのI/Oポートアドレス
……	……
GPE0_BLK	General-Purpose Event 0 Register BlockのI/Oポートアドレス(汎用イベント用)
GPE1_BLK	General-Purpose Event 1 Register BlockのI/Oポートアドレス(汎用イベント用)
……	……
P_LVL2_LAT	プロセッサがC2ステートに遷移するときのレイテンシ(マイクロ秒). 100より大きい場合はC2はサポートされない
P_LVL3_LAT	プロセッサがC3ステートに遷移するときのレイテンシ(マイクロ秒). 1000より大きい場合はC3はサポートされない
……	……
Flags	システムで固定の各種フィーチャーの有無を表すフラグ ●フィーチャーの例 キャッシュをフラッシュしてメモリのコヒレンシを維持するWBINVD命令のサポート プロセッサのC1ステートのサポート ボタンデバイスの実装形態 RTCアラームによるシステムスリープ解除のサポート ドッキングステーションのサポート

〔表4〕FACS構造体の主要フィールド

フィールド名	備　考
Hardware Signature	直近の起動時にBIOSによって記録され, S4(休止状態)からの復帰時にディスクに退避されたイメージの正当性評価に使用
Firmware_Waking_Vector	システムウェイク時にOSに制御が戻った直後, リアルモードで実行されるコンテクスト復元用ルーチンが格納された物理アドレス
Global_Lock	OSのコードと, SMIハンドラの双方によってアクセスされ得るハードウェア資源へのアクセスを同期させるグローバルロック

〔図9〕BIOSによる初期化

システム内に実装されたAMLインタプリタを呼び出して，ACPI BIOSから提供されたAMLコードを解釈し，ACPIタスクを実行していきます．

▶ DSDTのダンプ

インテルが無償で公開しているASLコンパイラ[2)]を使って，ユーザーが自分のPCのDSDTをダンプすることができます．

ASLコンパイラは，本来はBIOSエンジニアがASLコードをAMLに変換するために使うものですが，AMLを逆アセンブルする機能も追加され，Windows/Linuxどちらの環境からでも使用中のマシンのDSDTのダンプができるようになりました（**リスト2**）．

ASLマクロなどもきちんとデコードして，オリジナルに近いフォームに整形してくれるので解読が容易です．

● ACPIネームスペースに含まれるデバイス

▶ _HIDベースのデバイス

システムボード上には，デバイスの列挙とパワーマネジメントの手段が，自身のバス規格で定められていないレガシーなデバイスも存在します．

こうしたデバイスのコントロール手順を標準化するのもACPIの役割の一つで，対象となるデバイスはDSDTに記述され，ACPIネームスペース内で_HID（Hardware ID）オブジェクトにより識別されます．

リスト3（pp.187-188）の89～131行目に，サンプルとしてシリアルデバイスの定義例を示します．_HIDをもつデバイス配下に記述される，オブジェクトおよびコントロールメソッドについてコメントを付けてあります．

OSはACPIサブシステムを用い，ASLコントロールメソッドを介して_HIDベースのデバイスを直接列挙し，リソースを割り当て，OSが内部で管理しているデバイスツリーに組み込みます．

デバイスのパワーマネジメントも，同じくACPIがASLコントロールメソッドに記された手順を実行し，直接コントロールします．

▶ _ADRベースのデバイス

これに対し，PCI/USB/CardBus/IEEE1394など，自身のバス規格のなかでデバイスの列挙/リソースコンフィグレーションやパワーマネジメントの取り扱いが規定されているデバイスは，ACPIが直接管理する必要はありません．

プラグアンドプレイが汎用的にサポートされたバス規格では，どのデバイスがシステムに追加されるかあらかじめ予測することができず，おのずとACPIネームスペースの範囲外となります．

こうしたデバイスについては，OSがシステムワイドで統一なルールに基づき，標準バスの規定を実現するバスドライバと，バス上の各デバイス用に用意されたデバイスドライバを利用してハンドルします．

ただし，これらの標準バス規格に準拠したデバイスであっても，システムボード上に常駐し，その制御に自身のバス規格で定められた以外の，付加的な設定情報を必要とするものがあります．

PCIデバイスの割り込みルーティングの設定は，その代表的なケースになります．

そのほかにも，たとえば標準バスのホストコントローラが，PCIのPME#やUSBのリモートウェイクアップのように，自分が管理するバス上のデバイスから届けられたシステムスリープ解除のシグナルをACPIレジスタにルーティングするための設定情報なども，これに該当します．

このような付加的な設定情報を追加するには，まずそのデバイスをDSDTに含め，自身のバス上での位置関係を示す_ADR（Address）オブジェクトによってACPIネームスペース内で識別されるようにしておきます．

これにより当該デバイスの存在をACPIに知らせ，ACPIに制御を補ってもらうことができます．

そして，前述のPCI割り込みルーティングであれば，_PRT（PCI Routing Table）という割り込みルーティング情報を定義したテーブルを記述します．

リモートウェイクのケースなら，_PRW（Power Resource for Wake）を定義して，シグナルをルーティングするACPIレジス

〔リスト2〕iasl-hとiasl-dによるDSDTダンプのようす

```
// ここでは，Windows上からシステムのDSDTをディスアセンブルするようすを
// 紹介します
C:¥> iasl -h
     :
ACPI Tables and AML Disassembler:
                                    // ディスアセンブル機能に関係するオプション
  -d  [file]   Disassemble AML to ASL source code file (*.dsl)
  -dc [file]   Disassemble AML and immediately compile it
               (Obtain DSDT from current system if no input file)
  -g           Get ACPI tables and write to files (*.dat)
     :

C:¥> iasl -d

Intel ACPI Component Architecture
ASL Optimizing Compiler / AML Disassembler version 20030122
                                                        [Jan 22 2003]
Copyright (C) 2000 - 2003 Intel Corporation
Supports ACPI Specification Revision 2.0b

DSDT obtained from registry, 27178 bytes
  // Windowsの場合，ACPI BIOSの内容をレジストリにコピーしたものが
  //ディスアセンブルの対象
Table [DSDT] written to "DSDT_SAMPLE.dat"

Disassembly of DSDT
Pass 1 parse of [DSDT]
Pass 2 parse of [DSDT]
Parsing Deferred Opcodes (Methods/Buffers/Packages/Regions)
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
...........
Parsing completed
Disassembly completed, written to "dsdt_SAMPLE.dsl"
  // コメントやマクロの記述方法を除けば，内容はACPI BIOSに
  // 格納されているものと同一
```

タ内のビット位置を明確にします．

チップセットに統合された各バス規格のホストコントローラデバイスは，ほとんどの場合，上述のような理由によりDSDTに含めます．

また，ドッキングステーションの拡張ボード側に内蔵されたデバイスもDSDTに含めることが普通です．

最近では，USBルートハブやUSBポートをDSDT内に記述するケースもよく見かけます．

リスト3に，_ADRをもつデバイスの例として，Host-to-PCIブリッジ(16行目以下)と，その下にぶら下がるUSBホストコントローラ(35～83行目)を取り上げています．

ACPIの初期化

OSの起動過程で，ACPIサブシステムは，上述のACPIテーブルをロードしていきます．ロードしたテーブル情報は，内部で定めた構造にマップし，この後のOS側での利用に備えます．

また，ACPI対応システムであっても，ACPI非対応OSとの互換性などの理由でSMIとSCIの双方の割り込みメカニズムがサポートされている場合は，システムをACPIモードにする作業が発生します．

ACPIモードでない場合，プラットホームハードウェア上で発生したパワーマネジメント関連イベントは，OSからは透過的なSMIで通知され，OSから「見えない」モードでハンドルされます．

一方，ACPIモードでは，プラットホームハードウェア上で発生したACPIが管理すべきイベントが，SCIによって「ACPIイベント」としてOSに通知されるようになります．

このようなACPIの初期化の流れを**図10**に示します．

〔図10〕ACPIの初期化のフロー

● ACPIモードへの移行

この作業は，FADT内のACPI_ENABLEフィールドの値をSMI_CMDフィールドで指定されたSMIコマンドポートに書き込むことで実現されます．

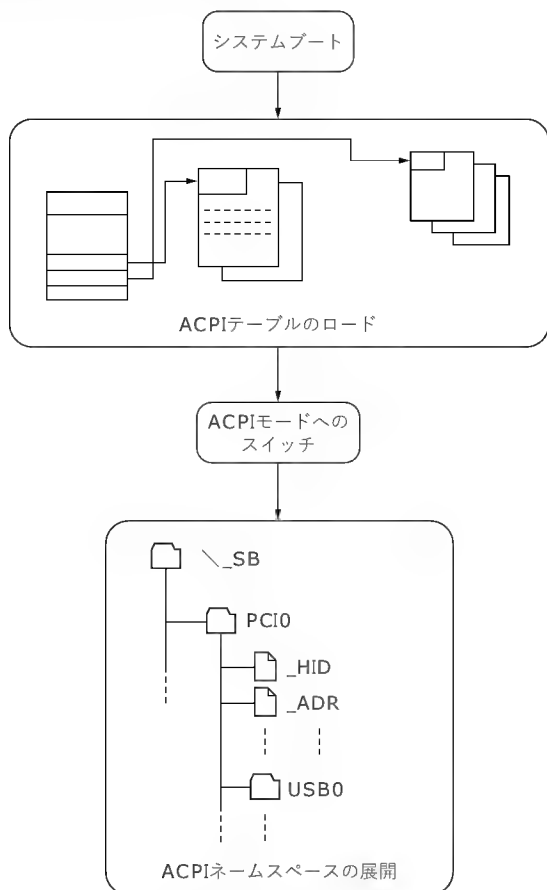
ACPIサブシステムは，FADT内のPM1a_CNT_BLKフィールドが指すPM1 Control RegistersのSCI_ENビットを確認し，正しくACPIモードに移行できたかを判定します(**表5**)．

SCI_ENビットがセットされていれば，システムはACPIモードにあり，ACPIイベントはSCIを発行する回路にルーティングされ，OSから「見える」モードでハンドルされます．

なお，SCIは原則として共有可能なレベルトリガの割り込みで，極性はアクティブローです．典型的なIA-32シングルプロセッサ環境で，割り込みコントローラが8259互換PICカスケード接続の場合，ほかのPCIデバイスとIRQを共有することもあります．

● ACPIネームスペースの展開

続いてACPIサブシステムは，ACPIネームスペース全体を解析し，ASLコード内に定義された各種オブジェクトやバッファを初期化して，ACPIタスクがAMLインタプリタによって実行できる作業環境を整えていきます．

AMLインタプリタ実行の用意ができると，ACPIサブシステ

〔表5〕PM1コントロールレジスタ

ビット	フィールド名	コメント
0	SCI_EN	このビットがセットされていれば，システムはACPIモード．プラットホーム上のACPIイベントはSCIとしてOSに「見える」モードで通知される
1	BM_RLD	プロセッサがC3ステートに落ちているときに，このビットがセットされていると，バスマスタ要求の発生によってプロセッサはC0に戻る
2	GBL_STS	write-onlyで，OS側がGlobal Lockをリリースする場合にセットする
……	……	……
10～12	SLP_TYPx	システムスリープの際に，実際に移行するシステムステートを示す値を書き込む
13	SLP_EN	システムスリープ移行のソフトウェア的な準備が完了した時点で，このビットをセットすることで，SLP_TYPxにプログラムしたシステムステートへ実際に落ちるハードウェア的なシーケンスが開始される
……	……	……

ムは，オブジェクトの評価やコントロールメソッドの実行を開始します．

このステージでは，OSはACPIを利用して，じつにさまざまな作業を行います．

ACPIネームスペースを母体にして，OSが内部で管理しているシステムワイドなデバイスツリーを展開する作業も，この段階で成されます．_HIDで識別されるデバイスは，OSがACPIを利用して直接列挙し，リソースを割り当て，OSのデバイスツリーに組み込みます．また，_ADRをもつデバイスに対して，その制御を補完するための準備をしていきます．

● オペレーション・リージョン・ハンドラ

ACPIでは，ASLコードからI/Oポートやシステムメモリ，PCIコンフィグレーション空間といった領域にアクセスするために，オペレーション・リージョンを定義し，これを経由して仮想的にハードウェア資源にアクセスします．

実際のハードウェアをアクセスするには，たとえばPCIコンフィグレーション空間内に定義したオペレーション・リージョンであれば，PCIバスドライバのような，対象領域のアクセスに関する知識をもったレイヤを利用する必要があります．

AMLインタプリタと，こうしたACPIサブシステム外部のレイヤとの間を取り持ち，ASLによるオペレーション・リージョンへのアクセスを可能にする仕事を受け持つのがオペレーション・リージョン・ハンドラです．

対象領域の性質によっては，初期化時にオペレーション・リージョン・ハンドラを登録し，ASLコードからのオペレーション・リージョンへのアクセスに備えておく必要があります．

以上，ここまで解説してきた事項を**リスト3**のサンプルコードをベースに整理しておきましょう．**リスト3**を見てください．

▶ PCIルートの取り扱い

まず16行目に，システムバスのスコープ内で，Host-to-PCIブリッジが，PCI0という名のデバイスとして定義されています．

そして，17行目の_HIDオブジェクトで，自分自身のPnP IDがPNP0A03（PCIバス）であることをOSに知らせています．

OS側のデバイスツリー内でのHost-to-PCIブリッジ＝PCIルートの扱いは，各ACPI対応OSの設計上のポリシーによるところです．

Windows XPのWDMデバイスツリーを例に取ると，ACPIドライバが管理するPDO（Physical Device Object：バス上のデバイス列挙の責任を負う）に，PCIドライバが作成したFDO（Functional Device Object：そのデバイス本来の機能を果たす）がぶら下がって，Host-to-PCIブリッジ用のデバイススタックを構成しています．

図11は，このようすをOpen Systems Resource, Inc.社のデバイスツリーユーティリティで表示しているところです．

デバイスツリーユーティリティは，Windows XPのDDKに付属しているほか，Open Systems Resource, Inc.社のサイト[3]から単独でダウンロードすることもできます．

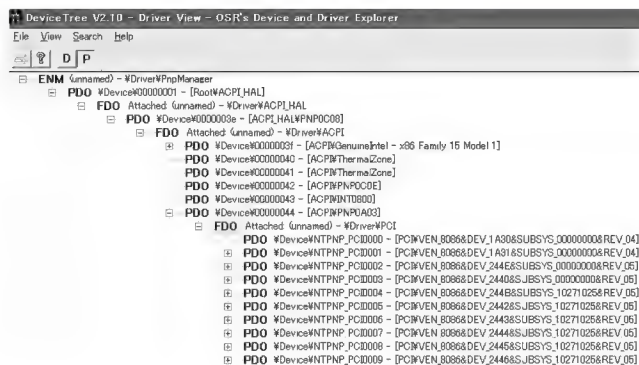
〔リスト3〕サンプルASLコード

```
  1:Scope(¥_SB) {
  2:  :
  3: Device (LNKA) {
  4:     Name (_HID, EISAID ("PNP0C0F"))
                                    // PCI 割り込みリンク仮想デバイス
  5:           :
  6:     Name (_PRS, ResourceTemplate () {
  7:       // PCI割り込みリンク仮想デバイスLNKAに割り当て可能なIRQ
  8:       IRQ (Level, ActiveLow, Shared) {3,4,5,6,7,9,10,11}
  9:     })
 10:     Method (_DIS, 0, NotSerialized) {...}
 11:     Method (_CRS, 0, NotSerialized) {...}
           // 現在割り当てられてIRQ
 12:     Method (_SRS, 1, NotSerialized) {...}
           // 有効なIRQをLNKAに割り当てるメソッド
 13: }
 14:  :
 15:
 16: Device(PCI0) {
 17:     Name(_HID, EISAID ("PNP0A03")) // PCI バス
 18:     Name(_ADR, 0x0) // デバイス 0, ファンクション 0
 19:     Name(_BBN, 0x0) // PCI バス 0
 20:
 21:     // PCI Routing Table
 22:     Name (_PRT, Package (0x06) {
             // このPCIルーティングテーブルには6個のエントリが存在
 23:
 24:         // PCI バス 0, デバイス 0x1D の INTA は,
 25:         // PCI 割り込みリンク仮想デバイス LNKA
 26:         // に現在割りあたっている IRQ へルーティングされる
 27:         //
 28:         Package (0x04) { 0x001DFFFF, 0, ¥_SB.LNKA,  0x00 },
 29:
 30:              :  // 以下, 省略
 31:     })
 32:
 33:         :
 34:
 35:     Device(USB0) {  // USB ホストコントローラ
 36:         Name(_ADR, 0x001D0000)
               // PCI バス 0, デバイス 0x1D, ファンクション 0
 37:
 38:         // USB ホストコントローラの PCI コンフィグ空間内で,
 39:         // オフセット 0xC4 から 0x4 の長さぶん, USBR という
 40:         // 名前のオペレーション・リージョン --- ここでは USB
 41:         // Resume Enable Register がマップされていると仮定
 42:         // --- を定義
 43:         //
 44:         OperationRegion(USBR, PCI_Config, 0xC4, 0x04)
 45:
 46:         // USBR のベースから 2 ビット分を RSEN フィールド ---
 47:         // ルートハブの 2 つのポートからの Wakeup Event を
 48:         // 通知する PORT0EN, PORT1EN ビットをマップ
 49:         // していると仮定 --- として定義
 50:         //
 51:         Field(USBR, DWordAcc, NoLock, Preserve) {
 52:             RSEN,   2,
 53:         }
 54:
 55:         // Power Resource for Wake
 56:         Name (_PRW, Package (0x2) {
 57:             // USB ポートからの Wakeup Event をルーティングする
 58:             // ACPI 汎用イベントレジスタのビット位置
 59:             //
 60:             0x03,
 61:             // USB ポートからの Wakeup Event によって S3 状態
 62:             // のシステムをウェイクアップすることが可能
 63:             //
 64:             0x03
 65:         })
 66:
 67:         // Power State Wake --- このデバイスからシステム
 68:         // スリープを解除できるようにするかどうか設定
 69:         //
 70:         Method (_PSW, 1, NotSerialized)
 71:         {
 72:             If (Arg0)
                   // 引き数が 1 なら, スリープ解除機能を有効にする
 73:             {
```

（次頁につづく）

〔リスト3〕サンプルASLコード(つづき)

```
 74:                Store (0x03, RSEN)
                       // USB Resume Enable Register の PORT0EN,
                                              PORT1EN ビットをセット
 75:            }
 76:            Else  // 引き数が0ならスリープ解除機能を無効にする
 77:            {
 78:                Store (0x00, RSEN)
                       // USB Resume Enable Register の PORT0EN,
                                              PORT1EN ビットをクリア
 79:            }
 80:        }
 81:            :
 82:            :
 83:    }
 84:
 85:    Device (ISAB) {  // PCI-to-ISA ブリッジ
 86:        Name (_ADR, 0x001F0000)
                // PCI バス 0, デバイス 0x1F, ファンクション 0
 87:            :
 88:
 89:        Device (COM1) {  // シリアル・デバイス
 90:            // デバイス識別のための情報
 91:            Name (_HID, EISAID ("PNP0501"))
                // 16550A 互換 シリアルポート
 92:
 93:            // デバイスのステータスに関する情報
 94:            Method (_STA, 0, NotSerialized) {...}
                // デバイスの有効/無効, 取り外されているかなど
                                               ステータスをレポート
 95:
 96:            // デバイスへのリソース割り当てに関するコントロール
                                                     メソッドその他
 97:            Name (_PRS, ResourceTemplate ()
                // 割り当て可能なリソースのいくつかの組み合わせを返す
 98:            {
 99:                // 互換性およびパフォーマンス上のいずれにおいても
100:                // 最適なリソース割り当てパターン
101:                //
102:                StartDependentFn (0x00, 0x00)
103:                {
104:                    IO (Decode16, 0x03F8, 0x03F8, 0x01, 0x08)
                                    // IO ポート 0x3F8 から 8 バイト
105:                     IRQNoFlags () {4}  // IRQ 4
106:                }
107:
108:                // 互換性の点からは受け入れ可能で, パフォーマンス
109:                // 上は最適なリソース割り当てパターン
110:                //
111:                StartDependentFn (0x01, 0x00)
112:                {
113:                    IO (Decode16, 0x02F8, 0x02F8, 0x01, 0x08)
                           // IO ポート 0x2F8 から 8 バイト
114:                     IRQNoFlags () {3} // IRQ 3
115:                }
116:
117:                     :
118:
119:                EndDependentFn ()
120:            })
121:            Method (_CRS, 0, NotSerialized) {...}
                  // 現在割り当てられているリソース情報を返す
122:            Method (_SRS, 1, NotSerialized) {...}
                  // リソースを実際に割り当てる
123:            Method (_DIS, 0, NotSerialized) {...}
                  // デバイスを無効にし, リソースを開放する
124:
125:            // デバイスステートの設定手順を伝えるコントロールメソッド
126:            Method (_PSC, 0, NotSerialized) {...}
                  // 現在のデバイスステートを返す
127:            Method (_PS0, 0, NotSerialized) {...}
                  // デバイスステートを D0 にセットする
128:            Method (_PS3, 0, NotSerialized) {...}
                  // デバイスステートを D3 にセットする
129:                :
130:                :
131:        }
132:            :
133:            :
```

〔図11〕DeviceTree.exeのPCI FDOの部分

18行目の_ADRは，PCI0デバイスのPCIバス上でのアドレスを表現しています．Host-to-PCIブリッジなので，アドレスはデバイス0，ファンクション0です．

19行目の_BBNは，ホストバスに複数のHost-to-PCIブリッジが対等に接続されたマルチPCIルート構成のシステムにおいて，BIOSが割り当てたシステム内で一意のPCIバス番号です．

▶ PCI割り込みルーティング

このサンプルでは，システムバス直下の3～13行目に，LNKAというPCI割り込み用のIRQをプールする仮想デバイスが定義されています．ここでは割愛していますが，通常はこのようなPCI割り込みリンク仮想デバイス(リンクノード)が複数定義されているはずです．

PCIの割り込みは，レベルトリガで共有可能といっても，OSの割り込み処理の機構によっては，あまり多数の割り込みがチェーン化していると，パフォーマンス上のペナルティも無視できません．

そこで，OSはパフォーマンスへの影響などを考慮しながら，PCI割り込みリンク仮想デバイスへ，ACPIコントロールメソッドを介して実際のIRQを割り振っていきます．

PCI0配下の_PRT(22行目)では，PCIバス0上に存在する各デバイスのINTA～INTDを，どのPCI割り込みリンク仮想デバイスへルーティングするかを定義しています．サンプルでは割愛しましたが，やはり通常はこうしたエントリが複数個きちんと定義されていて，デスクトップのPCI拡張スロットにも対応できるようになっています．

このようにして，ACPI対応システムでは，従来BIOS ROMに含まれるシグネチャ“$PIR”のPCI割り込みルーティングテーブルで実現していたPCI割り込みのルーティングを，ACPIによる方法で置き換えています．

▶ USBデバイスによるリモートウェイクアップ

次に，PCIバス0上に存在し，_ADRベースでUSB0という名前で定義されているUSBホストコントローラについて見ていきましょう．

36行目の_ADRにより，自分はPCIバス0，デバイス0x1D,

ファンクション0であることを表現し，次いで44行目ではPCIコンフィグ空間内のオフセット0xC4から4バイトを，USBRという名前のオペレーション・リージョンとして定義しています．

このサンプルは，インテルのICH系統のチップセットを参考にしていて，USB UHCIホストコントローラのPCIコンフィグ空間のオフセット0xC4〜には，USB Resume Enable Registerがマップされていると仮定しています．

51〜53行目では，今度はUSB Resume Enable RegisterをマップしたUSBRオペレーション・リージョン内のベースから2ビット分をRSENフィールドとして定義しています．

この二つのビットは，USBルートハブの二つのポートからのWakeup Event通知を有効にするPORT0EN，PORT1ENビットをマップしていて，これらをセットすることにより，リモートウェイクアップの通知が可能となります．

56行目からの_PRWでは，まず最初のエレメントで，USBホストコントローラに到達したリモートウェイクアップシグナルをルーティングするACPI汎用イベントレジスタのビット位置を明確にしています．

_PRWの二つめのエレメントは，そのデバイスからのWakeup Eventによって解除可能な，もっとも深いシステムスリープステートを示しています．ここでは値が0x3なので，USB0の下に接続されたUSBインプットデバイスのリモートウェイクアップシグナルで，S3システムスリープを解除できることがわかります．

70行目の_PSW(Power State Wake)は，デバイスが備えるシステムスリープ解除機能を有効/無効に設定するメソッドです．

引き数が1なら，スリープ解除機能を有効にします．なお，サンプルでは，USB Resume Enable RegisterのPORT0EN，PORT1ENビットをマップしたRSENフィールドに0x3をストアして，両ビットをセットしています．

引き数が0なら，スリープ解除機能を無効にします．サンプルでは，今度はRSENフィールドに0x0をストアして，両ビットをクリアしています．

▶ _HIDデバイスの列挙

続いて，シリアルデバイスを例にして，_HIDベースのデバイスをOSが列挙するようすを見ていきます．

89行目を見てください．このサンプルでは，シリアルデバイスはチップセット内蔵のPCIバス上に存在しているPCI-to-ISAブリッジの下にぶら下がっていることを想定しています．

この物理的なトポロジを反映して，ACPIネームスペース内では，シリアルデバイスCOM1が，フルパス¥_SB.PCI0.ISAB.COM1に位置しています．

ACPIネームスペースのパス表記は，見てのとおり，非常にシンプルです．上の例では，仮想的なシステムバスの下にHost-to-PCIブリッジを表すPCI0が置かれ，このPCIバス上にPCI-to-ISAブリッジであるISABがあり，シリアルデバイスCOM1はその下にあることが表現されています．

またASLコード内で，各オブジェクトのスコープがはっきりしているときには，相対パス表記も広く使用されます．

さて，すでに述べたように，_HIDで識別されるデバイスはOSによる列挙を必要としています．

この例では，OSは，ACPIを利用してデバイスを列挙した後，127行目の_PS0メソッドを実行して，デバイスステートをD0にセットします．

_PS0メソッドは，シリアルデバイスをD0にセットするためのレジスタアクセスをラップしているので，ACPIとしては，_PS0メソッド内の各ステートメントさえ実行していけばよいことになります．

D0にセットするのと同時に，ACPIは97行目の_PRSオブジェクトを評価して，デバイスに割り当て可能なリソースの組み合わせの情報を得ます．

サンプルにあるように，ACPI BIOS側では，リソース割り当てパターンに優先度を付けてレポートすることができます．

OSは，これらの中から，ほかのデバイスと衝突のないものを選択し，122行目の_SRSメソッドを実行して，実際にリソースをデバイスに割り当てます．

_SRSメソッドに，ACPIで規定されたフォーマットで，設定したいリソースの情報を渡しさえすれば，あとは_SRSメソッド内の各ステートメントを実行することにより，リソースの設定が完了します．

このように，デバイスに対して何かを設定するタイプのコントロールメソッドは，基本的にレジスタアクセスをラップしているという点で，類似のしくみになっています．

OSは，最後に94行目の_STAメソッドを実行して，デバイスのステータスが適切に設定されているか確認しておきます．

おわりに

今回は，ACPIの基本事項のうち，代表的なACPIテーブル群と，システム起動時のACPIサイドの初期化プロセスまでを見てきました．

ここでしばらく，「休止状態」に入って英気を養い，次回は，ACPIイベント機構の解説から，本稿を「レジューム」したいと思います．

参考文献，URL

1) ACPIオフィシャルサイト，http://www.acpi.info/
2) Intel ASLコンパイラ/AMLディスアセンブラ，
http://www.intel.com/technology/IAPC/acpi/downloads.htm
3) Open Systems Resource, Inc. デバイスツリーユーティリティ V2.10,
http://www.osr.com/files/devicetree_v210.zip

あだち・けんいち

● アトムの時代がはじまった

2003年の4月初旬，世界最大のロボット博覧会であるROBODEX2003が横浜で開催された．ホンダのASIMOが歩いて会場に登場し，ソニーのAIBOは新体操のリボンを披露していた．じつは今回のROBODEXは，アトムの誕生にちなんで行われたヒューマノイドロボットのデモンストレーションだったようだ．それもあって，プレス会場では，アトム誕生の直前までを見せてくれた．誕生の直前というのがちょっと残念だったが，ROBODEXの最終日がアトムの誕生日の前日だということで，誕生するところまで見せて歴史を変えてしまってはいけないという日本人らしい配慮だろう．

そんな日本が，世界でもっともヒューマノイドロボットの研究開発に熱心な国らしい．それは，「アトム」を愛した国だからだとロボット開発の第一人者たちは声をそろえていう．それほどアトムの影響は大きい．

しかし，ヒューマノイドロボットがアトムの域に達するのはまだまだ先で，30年から50年を目標にしたいと専門家は語っている．だが，技術の進歩は概して予想より早い．新しい発明や発見が技術の進歩を急激に進めたり，偶然が大きな障害を一気に解決させていくということを，過去の科学技術の発展の歴史が物語っている．アトムが地上に現れるのも，もしかしたらそんなに遠いことではないのかもしれない．

ROBODEX会場の続きは，翌日の兵庫県の手塚治虫記念館で見ることができたそうだ．午前10時半になったとき，ベートーベンの交響曲第5番「運命」が流れ，展示されていたアトムが地上に生を受け，起き上がってメッセージを伝えると，まわりから歓声が上がったそうだ．最初，アトムが話す予定はなかったが，多くの人から要望が寄せられていたため，急遽メッセージを伝えることにしたという．

● ヒューマノイドは神への冒涜(とく)だ！

ところで，ヒューマノイドは神への冒涜だと非難する人がいる．その主旨は，次のようなことだ．

神は自らにそっくりな人間を創造した．しかしサルトル，ニーチェ以降，人間は神の存在を否定し，自然環境を破壊，遺伝子を操作し，人間を神とする宗教までをも生み出した．傲慢になった人間は，今度は自らを最終的なゴールとしたロボットを作ろうとしている．これは，神への冒涜以外の何ものでもない．

哲学者然とした方々のいうことはよくわからないし，神への冒涜であるかどうかは，この際どうでもいいような気もするが，開発が無制限に進むことに対する警鐘とも受け取れば，公害を見てきた世代としては，理解できなくはない．

その一方で，ヒューマノイドロボットは不要だと断言する人もいる．その論理によれば，ヒューマノイドは投資効果に見合うだけの働きが期待できない．それでも騒がれているのは，単に希少な見世物としてのみ価値があるからだ．たとえ将来，大量生産され，安価で誰でも手に入る『道具』になったとしよう．道具にするためにロボットを使うならば，道具をロボットにしたほうが使い勝手がよく，安上がりだ．たとえば，ロボットを介護ベットで働かせるなら，介護ベッドをロボットにしたほうが合理的だということだ．

この種の議論は，筆者が知る限り，20年以上の間繰り返されてきたような気がする．しかし，この論理は誤っている．たしかに人間は万能の神を超えることはできないが，ある肉体的，精神的な特殊技能において，神にまでも近づく最高の技能――ある人はそれを芸術的と呼ぶかもしれない――を身に付けることができる．その最高の技能をロボットに実現させ，より多くの人々に，より長い間，人間をサポートすることができれば，人間の幸せを実現するために大きな価値が生まれるはずだ．

しかし，条件がある．人間らしさをもってこそ，ヒューマノイドの意義がある．人間と一緒に行動し，自らの感情を積極的に人間に伝え，人間の気持ちを理解することで，はじめて人間と共感でき，本当の意味で人間をサポートできるからだ．

● 人間らしさの価値はどこにある

印鑑ロボット，寿司ロボットなど，単純作業を効率的に繰り返すだけならば，産業用ロボットでも十分だ．人間が寝ている間も，ただひたすら効率的に仕事をこなして結果を上げればいい．SP，ガードマンロボット，救助ロボットなども，別にヒューマノイドでなくてもいいかもしれない．目的の機能を実現することが最優先だからだ．

しかし，人間の輪の中に入るとすれば，人間との会話による滑らかなコミュニケーションが必要だ．産業ロボットでは，使用する人間側に一方的にストレスが溜まる．物いわぬ機械が思

うように動かないときに蹴飛ばしたくなる，あの気持ちだ．

ネコ型ロボットのドラえもんは，のび太をあるときはやさしく見守り，あるときは厳しく教育している．のび太の家族も，そしてTVを見ている我々もこれを自然に受け止めていられるのは，ドラえもんが人間らしさがある超人（？）だからだ．

生涯の伴侶を失った一人暮らしのお年寄りや，重度の身体障害者をあらゆる面でサポートできる介護ロボットがあれば，その価値は高い．人生に疲れた自殺志願者には，精神科医なみに心の闇を打ち砕いてくれる，お友達ロボットが必要かもしれない．辛抱強く，いつでもどこででも付いて来てくれ，一緒に笑い，そして泣き，話し相手になってくれる．そのため，ヒューマノイドロボットの人間らしさに大きな価値がある．

これを実現するためにも，心理学者，言語学習，医学者，介護研究者，教育家などの各方面の専門家との共同研究も推し進められるべきだろう．

● ヒューマノイドエンジニアロボット

思うに，技術者ロボットというのもあるかもしれない．たとえば，最新技術や技法をす早く学習し，技術者にレクチャしたりする．初心者や年をとった覚えの悪い技術者にも，辛抱強く教えることができればいい．これが人間だったら，申し訳ないと思って遠慮してしまうこともあるが，人間でないわけだから，いくらでも聞ける（このへんに「人間らしさ」の微妙な線引きが必要なのだが）．

開発に直接参加して大きな力になるかもしれない．会話インターフェースを備え，人間と打ち合わせしながら数倍のスピードで開発を進めることができると大きな価値がある．

そうなると，ロボットが新しい技術の空洞化を生み，人間が技術をなくしてしまう問題がおきるかもしれない．しかし，パソコンが普及して，漢字を書けなくなったからといって，パソコンの文化を否定する人はいないだろう．なぜならば，人間はパソコンを利用して，さらに高度な能力が要求され，それを実現できるように進化していったからだ．

ロボットには，プロトタイプ開発やコーディング，テスト，1次サポートなどの労働集約的作業をやってもらえばいい．人間はマーケティング，企画，設計，顧客サポートなどに，より高度な作業に力を入れることができるようになるはずだ．

● 日本のロボット工学に金字塔を

最後に一つ，大きな問題が残る．それは，価格の高さである．どんなに大量生産されても，やはりかなり高価なものになるだろう．だがリース契約にすれば，人間に給料を払うのと同じくらいの金額になるかもしれない．また，文化的，福祉的分野においては，公共支援の可能性もあるだろう．将来の公共予算は，一般家庭や企業おいて人間と一緒に暮らし，技術や生産性を高め，ゆとりをもたらすヒューマノイド型ロボットにしてもらうべきだ．

筆者には一つの夢がある．ヒューマノイドロボット工学を制して，日本が再度世界のトップに立つことだ．最近，日本もノーベル賞をとることのできる科学者が多く登場するようになった．世界一といわれる日本のロボット工学の研究者の中から，多くのノーベル賞の受賞者を出し，金字塔を打ち立ててほしい．そんなに大きな夢物語ではない気がするのだが．

あさひ・しょうすけ　テクニカルライター
イラスト　森　祐子

Engineering Life in

ビジネススキルを修行しながらエンジニアを続ける

▌今回のゲストのプロフィール
羽田直樹(はだ・なおき)：1995年，電気通信大学卒業後，インターネット関連のプロジェクトのために渡米した．その後はWebのバックエンドを専門に扱う．仕事もビジネスも友達も妻も，インターネット経由で獲得．Adobe Systems, Inc.では，1998年よりインターネット関連の製品に関わり，最近Photoshopチームに移籍．週末はオフロードバイクで野山を駆け回る．『金持ち父さん，貧乏父さん』と『非常識な成功法則』に感化され，エンジニアを続けながらビジネスと投資のスキルも修行中．メールアドレスは，naoki@hadaseicha.com

☆ 日本のバイト先からの紹介で渡米する

トニー 本職以外にもいろいろと活動されているそうですが，まずはアメリカに来られたきっかけからお願いします．

羽田 日本でプログラマのバイトをしながら夜間の大学に通っていたのですが，卒業したらアメリカで働けたらいいなぁ，と仲間と話していたのがきっかけです．私はバイクが好きなので，天気がよくて高速道路の多いカリフォルニアのような環境が良いな，とか思っていました．また，コンピュータが発明された国に行くことも大事だと思っていました．

そこで移住する方法を調べようと市役所や都庁に問い合わせてみたりもしましたが，あまりわかりませんでした．今度は，いろいろな著書を調べているうちに，どうやらビザをスポンサーしてくれる雇い主が必要であることに気がついたのです．たまたまバイトの面接に行った会社でアメリカに関連会社があるから協力できるといってくれました．そして夏休みにとりあえず下見を兼ねて一度行ってみて，1年後にそこの知り合いの紹介でアメリカに引っ越しました．ロサンゼルス近郊のニューポートビーチです．でも，結局来てすぐわかったのですが，その会社はリストラ中で，いちばん肝心な就労ビザのサポートもどうなるかわからず，非常に不安でした．

トニー なるほど．自分の力でいろいろと調べたりして来られたわけですね．ロサンゼルス方面からシリコンバレーには？

羽田 同じくロサンゼルス近郊のパサデナとかリバーサイドでも仕事を少しした後に小さな会社に入り，会社ごとシリコンバレーに移ったのです．引っ越してみたところ気候が全然違い，ロス方面と比べてかなり寒いので驚いた記憶があります．

シリコンバレーに来て6年半になりますが，最初は知り合いがまったくなく，インターネットでいろいろと友達を増やしていきました．仕事はnewsgroupの仕事カテゴリとWebで検索し，ヒットした250件ぐらいの募集に何も読まずに自分のレジュメを送りました．友達のほとんどはメーリングリストで知り合いました．Adobeの仕事は，趣味のバイクやゲームのメーリングリストで知り合った人からの紹介でした．

トニー う～ん，なるほど……インターネットで徹底的にネットワークを広げているようですね．やっぱり自分をサポートしてくれたり知らないことを教えてくれる人をうまく周りに見つけるのは大事ですね．

☆ Adobeでの仕事について

トニー 小さい会社からAdobeに入ってどうですか？　Adobeは，現在シリコンバレーを代表する会社になっていますよね．たしか全米でもHPなどと並んで福利厚生がしっかりしているので「働きやすい会社」にランクインしていると覚えています．

羽田 そうですね，落ち着いた感じがあります．入る前の会社が社長と二人だけだったりと，不安定な感じだったこともあるかもしれませんが．また，グループが一致団結して仕事をしていると感じられ，非常に仕事はやりやすいです．わからないことなどは，バグデータベースや社内メールでやり取りをしたりしますし，オフィス[注1]へ聞きに行くことも多いです．

トニー 現在のお仕事では具体的にどんなことを？

羽田 Adobeでは，1年ほどQE(Quality Engineer，品質エンジニア)をしていまして，その後，おもにWeb関連のサーバスクリプトの開発を行うグループに移りました．C++やJavaも少し使ったのですが，VBScript，Jscript，PHP，JSPなどのスクリプト言語が多かったですね．また，Adobeの製品を拡張するためにJavaScriptベースの言語をよく使いました．でも，型宣言の甘い言語はデバッグがとてもたいへんなので好きになれませんね(苦笑)．

現在は，違うチームでWhitebox QEの仕事……プログラムも書くQEをやっています．ここではJavaScript，VB，AppleScriptを使っています．また，テストファイルを作成するためにPHPを利用することがあります．ツール系では，Perforce，Visual Studio，秀丸，FileMaker Proをよく使っています．

トニー なるほど．やはり最近はスクリプトを使う傾向が多いのですね．そのほかに今後どんな分野に興味がありますか？

羽田 Webのサーバサイドのプログラミングに興味があります．WDSLベースのオーサリングなどですね．サーバサイドの分野としては大きくなりましたが，やはり自分の作ったプログラムをいろいろな人に見せられる満足感が得られるのかもしれません．今後アメリカではブレイクするワイヤレスのクライアント-サーバ系も面白いと思っています．

☆ 起業と本当にやりたいこと

トニー それでは今後またスタートアップとかに行くとか？

羽田 そうですね，スタートアップに1人の従業員として入ることにはもう魅力を感じられません．自分ががんばっても会社の都合でリストラなどがありますから．また，ストックオプ

注1：Adobeはサンノゼ市のダウンタウンに位置し，シリコンバレーでは珍しい高層ビルに入っている．オフィスは，1人1部屋の個室がほとんどである．

ションや給与面でも起業するほうが割が合うと思うのです．従業員だとやはり従業員の給与ですから．だから，ぜひ起業して経営する側にまわってみたいと思っています．

トニー なるほど．たしかにシリコンバレーでは大きく二つに分かれると思いますね．スタートアップが好きな人と大手企業でしっかり仕事をしたい人と……あまりまん中がないですね．

羽田 じつは私にはもう一つ課題があります．実家は製茶業をしています．30歳になるまでコンピュータの仕事をして，その後は実家の仕事を継ぐ約束を父親としていました．今は31歳なので父親との約束を破ってしまっています．それでなんとかしたいと考えていました．

お茶の業界も変化があるわけで，海外でお茶をやっている大きなメーカーもあるし，家族経営的にやっているところは縮小しています．自分は，これまではハイテク業界でそこそこやってこれましたが，それもサラリーマンだし，家業経営とまた違うわけです．そして2001年の夏に読んだ本で『金持ち父さん，貧乏父さん』があり，この本にかなり感化されました．結論としては，自分には，財務的知識（Financial Literacy）が足りないと感じました．それで自分で株式投資，不動産などをいろいろと試してみました．まあ，おかげで企業のバランスシートなども読めるようになりました．

トニー 財務知識は起業にはたいせつですよね．まあ，エンジニアだと，それほど難しくなくピックアップできますよね．

羽田 しかし実際は，家業を手伝えるとか家業を大きくするための自信にはなかなかつながりませんでした．なにか空回りしていると感じ，自分にはさらに経営スキルが必要だと考えました．それで現在は会社に勤めながらサイドビジネスをやっています．これは，ハイテクとは関係のない仕事なのですが，これを少しずつやりながら経営スキルを身に付けようと思っています．もちろん技術的なスペシャリストになって，それをベースにして起業というルートもあるのですが，自分の空き時間でできることとか，他のバランスを考えて現在のサイドビジネスが自分にはもっとも適していると感じています．

トニー 感覚は理解できますね．なにか自分でやりたいけれどどこからスタートしてよいかわからない……．技術専門でいくか，経営でいくか？　私の場合はスタートアップにいきなりエンジニアから経営のほうに入りました．起業家で声をかけてくれた人がいたのですが，私から1人の従業員のエンジニアとして入るのでなく，経営サイドに入りたいと提案したのです．それで入ったのですが，とにかくわからないことばかりでした．幸いパートナーがいて，二人で簿記の付け方からビジネスプランの書き方とか経営に必要な知識の研究と勉強をしました．全部見よう見まねでやったという感じでした．その日の午前中に勉強したことを午後には平気で10年ぐらいやっているような顔をして投資家達に説明をしたりしてね……（苦笑）．冷や汗モノもあったのですがすごく身につきました．1人で研究・勉強しなければならないこともたくさんあるのですが，幸い周りにお手本や教えてくれる人もいたので，いろいろ人にサポートをしてもらったと思います．起業の知識・ノウハウはスタートアップには濃くありますから，給料をもらいながらビジネススクールに行った感じですね．

羽田直樹氏

羽田 スタートアップはシニアな人しか雇わないというイメージがあったのですが，いろいろなやり方があるのですね……なかなか面白い話です．現在のAdobeでの仕事はとても気に入っているので，それをやりながら何とかビジネススキルを磨いていきたいと思っています．ネットワークを広げたりして仲間を増やすこともたいせつですし．最終的な目標は，ハイテク企業を立ち上げて上場させてみたいですね．それでジェット機2台ぐらい所有できる一流企業に育てたいですね（笑）．あとは，趣味でやっているオフロードバイクがあるのですが，シリコンバレーのすぐ南にあるHollisterに，バイクに乗るパークがあります．1日4ドル出せば乗れるのですが，一回りするのに1時間強ぐらいかかる規模なので，日本では考えられないほど充実しています．こういうみんなが気軽にオフロードバイクを楽しめるパークを世界中に作りたいと思っています．

対談を終えて：

羽田 この対談は，自分の来た道を振り返る機会を与えてくれた．プログラミングは大好きだが，将来の自分の姿は別のところにあると感じる．自分の空き時間にそれに挑戦できる環境があるのはやっぱり幸せだ．昔は大好きだったTVゲームも最近遊ばなくなった．これは，ルールがわかってくると，現実はTVゲーム以上に面白いからか．

トニー 羽田氏は，本業のソフトウェアエンジニアリング以外にじつに多くの活動をされている．家業と自分の起業のためにやっているサイドビジネスや，英語のスピーチ向上のためにToastmasters[注2]など，かなり積極的かつ幅広い活動をされている．対談ではアメリカでの税法や小規模ビジネスの話など，筆者にとっては興味深い話が出たが，本誌の趣旨と離れているので残念ながら割愛した．今後の発展がじつに楽しみな若手エンジニアである．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

注2：トーストマスターズクラブ：スピーチなど，人前で話す技術を向上させるための非営利団体のグループ．

HARD WARE

●32ビットRISCマイコン

M32176グループ

- 最大512Kバイトの大容量フラッシュメモリを内蔵した32ビットRISCマイコン．書き込み時間は512Kバイトで約7秒．
- フラッシュメモリの書き換え温度範囲は，－40～125℃の広域温度範囲に対応したことで，マイコンの動作中や動作終了直後の書き換えが可能．
- 自動車用途の従来品「M32171グループ」の機能強化版で，ファンクションおよびピン配置が互換であるため，従来システムの高機能化，高性能化が容易．
- 電源降圧回路の搭載により，3.3または5.0Vの単一電源動作とtyp.65mA（5V，40MHz動作時）の低消費電流化を実現．
- 部品点数の削減が可能で，システムの低価格化，低消費電力化が図れる．

■（株）ルネサス テクノロジ
サンプル価格：¥5,000
TEL：03-5201-5211

●32ビットRISCマイコン

MB91302

- SDRAMインターフェースをはじめ，SRAM，非同期型ROM（EEPROM，マスクROM），フラッシュメモリ，FCRAMなどのインターフェースを搭載．
- 各種メモリとのダイレクト接続が可能となり，データ処理速度の向上や部品点数削減によるLSI設計の簡易化，低コスト化を図れる．
- データをCPU部に取り込まずに，外部入出力回路とメモリ間でダイレクトにデータ転送を制御する専用回路，高速DMACを搭載しているため，従来に比べ約2倍の転送レートを実現．
- プロセステクノロジは，CMOS 0.25μm．
- 動作電圧は，3.0V～3.6V（標準で3.3V）．
- 各4KバイトのROM，RAM，キャッシュメモリを搭載．
- 最大動作周波数は，68MHz．
- 外部供給クロックは17MHz，PLLは4逓倍．
- おもな機能として，DMAC，ICU，PPG，タイマ，リロードタイマなどを搭載．
- パッケージは，QFP 144ピンで供給．

■ 富士通（株）
価格：¥1,200
TEL：042-532-1397
E-mail：edevice@fujitsu.com

●統合コミュニケーションプロセッサ

RC32365 Interprise アクセス・プロセッサ

- 最大150MHzで動作する32ビットMIPS CPUコアのCPU．
- SDRAMメモリコントローラ，業界標準MIIインターフェースをもつ2系統のEthernet MACを内蔵し，広範囲なネットワーキングデバイスや装置と接続することが可能．
- バージョン2.2の32ビットPCIコントローラとバージョン2.1の16ビットPCMCIAインターフェースを内蔵．
- PCIとPCMCIAインターフェースは，WLANソリューションを含む広範囲なペリフェラルとチップセットへのシームレスな接続を提供．
- VxWorksやLinuxなどの一般的な組み込み向けOSと互換性を保持．

■ 日本IDT（株）
サンプル価格：$17.00（10,000個時）
TEL：03-3221-6726　FAX：03-3221-5456

●無線LAN用チップセット

MN103S640L-H

- 映像伝送標準形式のMPEG-TS信号に対応した入出力ポートを三つ備え，映像信号を3ストリームまで直接入出力することが可能．
- 無線映像伝送する際のエラー発生，データ再生により生じる時間的なずれをもとの画像信号に合わせて高精度補正する機能を内蔵．
- IEEE802.11a規格にしたがって，無線パケットごとに信号レベルを高速に検出し，アンテナを自動的に切り替える高速ダイバーシティ機能を搭載．
- 電波変動時の誤り率を従来比で1/10に低減．
- 高周波プロセスである0.25μm SiGe Bi-CMOSプロセスをRF ICに採用し，低消費電流で低歪，低雑音の5GHz高周波部ICを実現．

■ 松下電器産業（株）
サンプル価格：¥10,000
TEL：075-951-8151
E-mail：semicon-press@mrg.csdd.mei.co.jp

●FPGA

Spartan-3

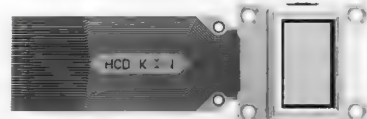
- 5万ゲートから500万ゲートにおよぶ高集積度を実現．
- 組み込みブロックRAMおよび分散型RAMの双方を組み合わせることが可能．
- 小さいチップ上に最大限のI/Oパッドを搭載するために，2重リングのスタッガーパッド配置を構成．
- 最高3300億MAC/秒というDSPアプリケーションに対して，低コスト性を備えた組み込みDSP機能を提供．
- 最高104個の組み込まれた18ビット乗算器と分散型RAMエレメントを装備．
- PCI 32ビット/33MHzおよびPCI 64ビット/33MHzを含む23種のパラレルI/Oスタンダード，シングルエンドとディファレンシャルシグナルの両方の信号形式に加え，他の通信およびネットワーキングのアプリケーション用として一般的なパラレルコネクティビティコアをサポート．

■ ザイリンクス（株）
価格：下記へ問い合わせ
TEL：03-5321-7740　FAX：03-5321-7762

●TFT液晶パネル

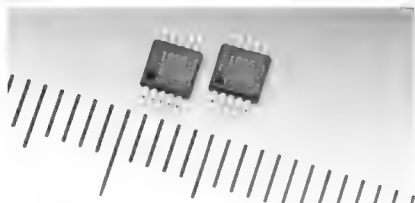
HTPS D4シリーズ

- 液晶プロジェクタ向け高温ポリシリコンTFT液晶パネル．
- 高精細加工技術により12μm画素ピッチを実現．同一サイズでの開口率を向上．
- 同一画素ピッチでも従来と同レベルの開口率を実現することで，液晶サイズの小型化を実現．
- 新平坦化技術，新液晶（2.5μm）を採用することで，なめらかな映像を実現．
- 従来と比較して，応答速度を25%改善．
- 高効率化により同一ランプでの輝度を向上．
- 同一解像度，同レベル開口率での液晶サイズの小型化およびハイコストパフォーマンスを実現．

■ セイコーエプソン（株）
価格：下記へ問い合わせ
TEL：042-587-7665
URL：http://www.epsondevice.com/

●モータドライバIC

LB1935T

- 低飽和駆動，低電圧動作，正/逆モータドライバを2チャネル搭載した，カメラ付き携帯電話などに用いられる2相ステッピングモータの駆動用IC.
- CPUからの信号によりモータを直接制御することができ，3本の信号線による2相ステッピングモータの2相励磁駆動用に適する.
- 出力は上側PNP＋，下側NPNのバイポーラタイプで構成されており，小型パッケージながら低飽和駆動，低電圧動作を実現.
- サーマルシャットダウン回路(過熱保護回路)を内蔵.
- MSOP10(3.0×4.9mm)の小型パッケージのため，基板実装の省スペース(面積比50%減，体積比70%減)を実現.

■ 三洋電機(株)
サンプル価格：¥100
TEL：0276-61-8107　FAX：0276-61-8730

●フラッシュメモリ

MBM29BS12DH
MBM29SL800TE
MBM29SL800BE

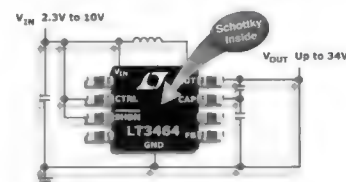
- 「MBM29BS12DH」は1.8Vの低電圧電源で動作する，携帯機器向けNOR型128Mビットのフラッシュメモリ.「MBM29SL800TE」「MBM29SL800BE」はBluetoothなどの無線通信モジュールや超小型携帯機器向けスーパーCSPを採用したNOR型8Mビットのフラッシュメモリ.
- 「MBM29BS12DH」は，電源投入後のアドレス固定からデータの出力までのイニシャルタイムは，最速で46ns．バーストモード時のアクセスタイムは8.5nsで，コンフィグレーションレジスタの設定により，システムに最適なバーストモードの選択が可能．バーストアクセス読み出しを実現するために，2ワード分のセンスアンプ数におさえることで，消費電力を低減化.
- 「MBM29SL800TE」「MBM29SL800BE」は，省スペース化が可能で，チップ単体に比べ二次実装信頼性が向上．動作電圧1.8Vで，アクセスタイム90nsの高速動作を実現.

■ 富士通(株)
価格：MBM29BS12DH ¥4,000
MBM29SL800TE/MBM29SL800BE ¥400
TEL：042-532-1416
E-mail：edevice@fujitsu.com

●マイクロパワー昇圧コンバータ

LT3464

- ショットキダイオードと短絡保護付き出力切断PNPトランジスタを搭載.
- 85mAスイッチを内蔵し，電流制限付きの一定オフタイムスイッチング方式を採用することで，最大34Vの電圧を供給.
- 3.6V入力から16V/8mAを供給可能なので，ハンドヘルド機器のバイアスアプリケーションに適する.
- 2.3～10Vの入力電圧範囲により，1セルまたは2セルのリチウムイオンバッテリやマルチセルのアルカリまたはNiMHバッテリに適する.
- バーストモード動作により，25μAの超低消費電流が可能で，シャットダウン時には0.5μA以下に低減される.

■ リニアテクノロジー(株)
サンプル価格：¥180～(1,000個時)
TEL：03-5226-7291　FAX：03-5226-0268
URL：http://www.linear-tech.co.jp/

●クロックシンセサイザ

CDC7005

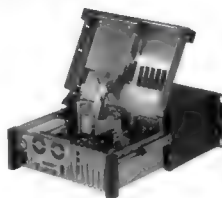
- VCXO(電圧制御水晶発振器)の位相を，リファレンスクロックに同期する機能を装備.
- 低雑音の位相周波数ディテクタ，精密チャージポンプ，プログラマブルデバイダ，OPアンプ，分周オプション付きの1：5差動クロックバッファなどを集積.
- 低い位相雑音特性を備え，A-Dコンバータ，D-Aコンバータ，シリアライザ，ASIC，DSPなどのリファレンスクロックへの精密な同期が必要なシグナルチューンデバイスに利点を提供.
- リファレンスクロックとして，3.5MHz～180MHzの範囲の入力と，それに同期させるための10MHz～800MHzのVCXOが必要.
- 低スキューの差動出力を5個提供し，10MHz以下の帯域でも最適なPLLのループ帯域幅の選択が可能．また，リファレンスクロック入力信号に含まれるジッタを除去する機能を装備.
- 内蔵のOPアンプは，ループ帯域幅を最適化するためのアクティブフィルタとして使用可能.

■ 日本テキサス・インスツルメンツ(株)
価格：$13.8(1,000個時)
FAX：0120-81-0036

●ATXシャーシ/セットアップPC

MPCシリーズ

- すべての外部ケーブル配線をフロント側に集中することで，すばやいメンテナンスが可能な組み込み向けPC.
- 背面にメンテナンスハッチが不要なため，収容される端末，装置全体を低コストでコンパクトに設計可能.
- メインユニット引き出し，トップカバー跳ね上げ機構のメンテナンス重視の設計のため，端末装置に設置したままメンテナンスが可能.
- 市販のATX仕様のマザーボードの組み込みが可能.
- 拡張ボードクランプ機構を標準装備.

■(株)コンテック
価格：オープン価格
TEL：03-5628-9286　FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●ICカードリーダ/ライタ

RR-01-01/RR-01-02
/RR-01-03

- RR-01-01は標準無線LAN，RR-01-02はエアロネット仕様無線LAN，RR-01-03はEthernetにそれぞれ対応.
- ISO/IEC15693の国際標準に準拠した，非接触ICカードリーダ/ライタで，市販のICカードやタグを利用したシステムの構築が可能.
- 無線LAN環境が構築されたエリア内であれば，自由に設置場所を移動することが可能.
- 無線LANのセキュリティが強化されたエアロネット仕様のモデルのほか，標準仕様の無線LANモデル，有線LAN仕様のモデルをラインナップ.
- 添付のユーティリティソフトにより，ユーザIDやパスワードなどの初期情報の書き込み，データのロックなどを画面上で行うことが可能.

■ 日本電気エンジニアリング(株)
価格：オープン価格
TEL：042-333-4673

HARD WARE

●レンズユニット

FMZ10

- FDK(株)と共同開発の，携帯電話やPDAなどに搭載する小型カメラ向けのレンズユニット.
- 光学2倍ズーム機能とマクロ撮影機能を搭載.
- 新開発の小型電磁アクチュエータを利用し，3群3枚で構成される非球面レンズの後群レンズのみをユニット内部でスライドさせることにより，レンズ部がせり出すことなくズームやマクロ撮影を可能にしている.
- 容積は1.1ccを実現.
- 樹脂レンズの採用と機構部の簡素化により落下などの衝撃に強い構造を実現.
- イメージセンサは，1/7インチCCDおよび1/7インチCMOSの両方に対応が可能.
- 駆動電圧は3V，駆動電力は0.45W.
- F値は2.8～3.9，撮影距離は3cm～∞.

■(株)マクニカ
価格：下記へ問い合わせ
TEL：045-470-9851

●メタチャネルモジュール

APM-750

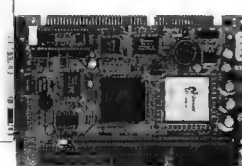
- 複数のシステム間でメモリを共有させることができる，ギガチャネルモジュール.
- 接続ケーブルにはRJ-45のメタルケーブルを使用し，通信速度は3.2Gbpsの高速を実現.
- 各ノードに搭載するモジュールのメモリをメタルケーブルでループ状に接続し，この間を通信データを載せたフレームを順次転送することで，高速データ転送を可能にする.
- 独自仕様のシンプルなプロトコルを採用しているため，異なるOS(VxWorks，Linux，Windows 2000/XP)やシステム間でもデータの共有が行える.
- 最大で128台の製品を接続することが可能で，CompactPCIモジュールなど，組み込みモジュールコンピュータの拡張用のPMCタイプとなる.
- 通信機能をハードウェアで実現し，任意のノード間への割り込みにも対応.

■(株)アバールデータ
価格：¥198,000
TEL：042-732-1030　FAX：042-732-1032

●CPUボード

ROCKY-512EV

- 台湾ICP社が開発した，ISAハーフサイズ(185×122mm)でローパワーCPUを搭載したCPUボード.
- CPUとメモリがオンボードで搭載されているため，インストール，組み込みが容易.
- 低消費電力のNS GX1 300MHzCPUは，環境温度60℃までファンレスで動作が可能.
- オンボードSDRAMは，64Mバイトおよび128Mバイトの2種類を用意.
- 拡張用に最大512MバイトのDIMMソケットを用意しているため，合計640Mバイトまでのメモリが搭載可能.
- ディスプレイは，CS5530Aチップセットを内蔵.

■(株)ケーメックス
価格：下記へ問い合わせ
TEL：03-5295-3111　FAX：03-5295-3123
E-mail：info@kmecs.co.jp
URL：http://www.kmecs.com/

●フラッシュメモリ用ICソケット

TSOP

- サンテスト社が開発したアミューズメント業界向け，TSOP56ピンフラッシュメモリ用ICソケット.
- ホルダ上面に大開口部を設け，シールを貼ったデバイスを目視で確認可能.
- ワンタッチロック方式により，操作性に優れている.
- 鉛フリー対応で，耐熱樹脂を採用.
- ICをふたにホールドすることで，足曲がりを防止.
- 実装機対応のためのエンボステーピング梱包.
- はんだ付け端子の肉眼による確認が可能.
- ソケットの挿抜を30回まで可能にし，リサイクルにも対応.
- ふたは2種類提供され，48/56ピンパッケージの装着が可能.
- ICの抜き取り治具を用意.

■加賀テック(株)
サンプル価格：¥350
TEL：03-5207-5661　FAX：03-5207-5664

●ICEツール

advicePOCKET

- 低価格市場をターゲットにし，OCDツールに特化したICE製品でJTAG対応ツールとフルICEシステムをリリース.
- 対応CPUはARM.
- JTAGインターフェースによる，簡単接続を実現.
- 1.8V～3.3Vの幅広い電源電圧に自動追従.
- 最高200MHzまでの分岐PCトレースに対応.
- サンプルのトレース&タイムスタンプは，256Kという大容量をサポート.
- 外部フラッシュメモリへの書き込みに対応.
- 外部トリガ入出力機能による，連携デバッグに対応.
- ホストへの接続は，USB対応.

■横河ディジタルコンピュータ(株)
価格：下記に問い合わせ
TEL：042-333-6222　FAX：042-352-6107

●プロトコル解析用測定器

IEEE1394Bデータアナライザ「IP1200」

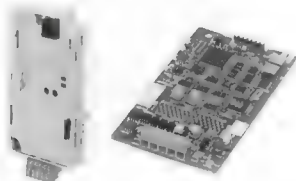
- IEEE1394b-2002に対応した，プロトコル解析用測定器.
- IEEE1394a-2000およびIEEE1394b-2002の双方に完全対応する，バイリンガル仕様.
- Webプラットホームを採用し，汎用ブラウザ上で動作するため，HMIとして使用するパソコンOSの種類を問わない.
- ネットワーク経由で遠隔地からの操作も可能.
- ディスプレイとキーボードを接続することで，スタンドアロン型のアナライザとしても利用可能.
- データの物理層だけではなく，AVC，SBP-2，IPv4Over1394の3種類の上位プロトコルの解析機能を標準装備.

■横河電機(株)
価格：¥2,900,000
TEL：0422-52-5556　FAX：0422-52-6624

●インバータ用DeviceNet通信ユニット/カード

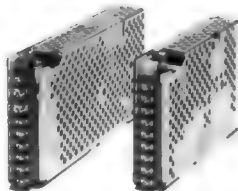
3G3MV-PDRT2 3G3RV-PDRT2

- インバータに搭載することで，インバータや周辺機器を監視し，設備の故障診断や故障予知に生かせる機能を搭載．異常が発生してから復旧までのダウンタイムを短縮することが可能．
- 定速運転中時，加減速中時のそれぞれの出力電流値に閾値を設定し，出力電流が閾値を超えると警告を発するワーニングトルク機能を搭載．
- 電力トレース機能は，モータの出力電流波形を150個までサンプリングすることが可能．
- サンプリング周期に応じて，出力電流値をインバータ内部に記憶させることができる．

■ オムロン(株)
価格：3G3MV-PDRT2 ¥39,000
3G3RV-PDRT2 ¥39,000
TEL：055-977-9025

●箱型標準電源

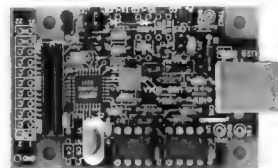
RTWシリーズ

- 独自の熱設計シミュレーション/実装技術と回路技術の最適化により高効率を達成した小型電源．
- 50Wタイプと100Wタイプをリリース．
- 厚さ寸法22mm(50Wタイプ)，重さ250gと，従来品と比較してサイズを50%小型化．
- 標準の1Uラックサイズ(44mm)にも余裕を持った実装ができ，電源の裏表両面からの取り付けが可能．
- 省スペース化，省資源化を実現しつつ，各国の安全規格，ノイズ規格，高周波電流規制，CEマーク適応など各規格に対応．

■ TDK(株)
サンプル価格：¥8,400(50Wシリーズ)
¥11,800(100Wシリーズ)
TEL：03-5201-7102

●USB開発キット

USB-004-DK

- FTDI社のUSB-FIFO変換LSI，USB-シリアル変換LSIを用いた開発キット．
- USB-004-FIFOとUSB-004-SERのセットに，デバイスを各1個(FT245BMとFT232BM)をセットにしたキット．
- USB-004-FIFOは，FT245BMを使用したUSB-FIFO変換基板．8ビットパラレルポートと，四つの制御信号でFPGA/CPLD基板，マイコンとの接続が可能．仮想COMポート(VCP)ドライバを用いて，アプリケーションからCOMポートとして操作が可能．
- Windows 98/Me/2000/XPに対応したドライバのほか，LinuxやMac用のドライバも用意．

■(有)ヒューマンデータ
価格：下記へ問い合わせ
TEL：072-620-2002　FAX：072-620-2003

●USB2.0リンクケーブル

瞬転U2LC-480

- パソコン間をUSBで直結し，ファイル，データの転送が可能．
- USB2.0のハイスピード(480Mbps)対応で高速転送が可能．
- ASICをケーブルの中に組み込んだシンプルな構造で，ノートパソコンのデータ転送に適する．
- 重さ120gの軽量．
- プラグアンドプレイ対応で，簡単接続を実現．
- USB自身から電源を供給するタイプなので，電源が不要．
- USB1.1のホストでも使用可能．

■(株)コンパル
価格：¥6,980
TEL：03-3299-5751　FAX：03-3299-5059
URL：http://www.compal.to/

●無線プリントサーバ

KP-612air

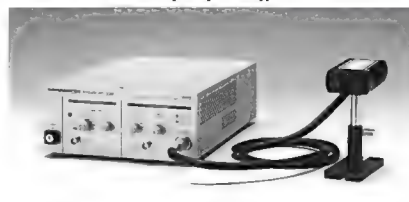
- プリンタに直接接続して簡単に無線ネットワーク化できる，1ポートマルチプロトコル無線プリントサーバ．
- 300Kバイト/sのスループット性能を実現しており，プリンタの性能を最大限に引き出すことが可能．
- 有線，無線の混在環境でも速度を気にせずに印刷可能．
- データ形式をバイナリタイプにも対応することで，印刷データのサイズを小さくすることができる．
- プリンタメーカ純正ドライバを使用した印刷も可能．

■(株)小松製作所
価格：¥24,800
TEL：045-441-2701
E-mail：lan_sale@komatsu.co.jp

●ライトパルサ

PLP-10

- LDを用いた超短パルス光源で，コントローラとレーザダイオードヘッドで構成．
- 400nm～1550nmまでの波長範囲で8種類のレーザダイオードヘッドを用意．
- 最大100MHzまでの高繰り返しが可能で，従来品と比較して出力光が強く明るくなり，光軸調整も簡単．
- POF光通信やギガビットEthernet用のファイバ周波数特性評価，受光素子の応答特性評価などで，タイミング調整が簡単となり，信号をとらえるための周辺機器も不要．
- 蛍光寿命測定用の励起光源として，データの取得時間が短くなり，より多彩な用途に使用することが可能．

■ 浜松ホトニクス(株)
価格：¥1,240,000～
TEL：053-452-2148　FAX：053-452-2139
E-mail：sales1@sys.hpk.co.jp

SOFT WARE

●プロセッサ開発キット

Niosエンベデッド・プロセッサ開発キット Cycloneエディション

- Niosエンベデッドプロセッサ Ver3.0，Cyclone EP1C20デバイス，Quartus IIデザインソフトウェア，SOPC Builderシステムデザインツール，およびソフトウェア群が含まれる．
- Cyclone EP1C20デバイスを搭載したCyclone開発ボード，1MバイトのSRAM，16MバイトのSDRAM，8Mバイトのフラッシュ，10/100BaseEthernetポート×1，シリアルポート×2，ソフトウェアデバッグ用Mictorコネクタ×1，拡張ヘッダ×2，電源およびダウンロードケーブル×1で構成．
- オンボードフラッシュメモリにプリロードされたリファレンスデザインとハードウェアおよびソフトウェアチュートリアルが含まれる．

■ 日本アルテラ（株）
価格：$995
TEL：03-3340-9415　FAX：03-3340-9487
URL：http://www.altera.co.jp/

●ネットワーク構成図作成用ユーティリティ

NetZoom

- Visio，PowerPoint，AutoCAD，Illustratorなどのソフトと同時に使い，プロフェッショナルなプロポーザルやプレゼンテーション用の資料作成が可能．
- NetZoomStencil，NetZoom，NetZoom Symbolの3種類があり，技術者あるいは用途により使い方が異なる．
- Cisco，IBM，SUNなど1,100以上のメーカと50,000点以上の機器のイラストをVisio，PowerPointなどのホストアプリケーション内にドラッグアンドドロップ可能．
- 12か月間イラストを更新ダウンロード可能なスタンダード版と，30日間用のベーシック版を用意．

■ アイ・ビー・エス・ジャパン（株）
価格：下記へ問い合わせ
TEL：046-234-9200
E-mail：press@ibsjapan.co.jp

●Linux製品開発ツールキット

RADVISION SIP開発ツールキット/RADVISION H.323開発ツールキット

- MontaVista Linux Professional Editionを使用したLinuxベースの製品の開発向けツールキット．
- RADVISION SIP開発ツールキット バージョン2.2は，オープンなオブジェクト指向型のアーキテクチャをもち，ハイレベルAPIからミドルおよびローレベルAPIまで複数の階層型APIをサポート．メッセージエンコーディング/デコーディング，トランザクションとコールマネージメントなどすべてのSIP固有の機能や，REFER，リプレース，PRACK，SUBSCRIBE NOTIFYなど多くのSIP拡張を含んだライブラリを装備．SDP，RTP/RTCPライブラリも装備．
- ITU H.323バージョン4.0をベースとした，RADVISION H.323開発ツールキットは，包括的なITU機能のセットと低消費メモリで高性能を提供．移植性が高く，すべてのネイティブおよび組み込みプラットホームに対応．

■ モンタビスタソフトウェアジャパン（株）
価格：下記へ問い合わせ
TEL：03-5469-8840

●プリント基板設計/製造の通販サービス

P板.com

- 2層，4層のローエンドな基板に特化し，1～50枚までの小ロット製造を専門とする，インターネットを利用したプリント基板の設計と製造サービス．
- 受注可能な基板の仕様を標準化することにより，効率化された製造フローを提携先の海外工場（韓国，台湾）と構築し，初期費用の完全無料化を実現．
- 標準的な仕様での設計納期は最短で4日，設計見積は受付完了から3時間以内に回答．
- 製造納期は最短で3日，製造見積は仕様入力時に即時自動回答．
- 標準仕様は，ピン数200ピン，外形サイズ70×90mm，層数2層，枚数5枚，指定回路図CAD（D2CAD，PROTEL，ORCAD）データでの受注．
- 品質と納期の管理は，各国で専属契約を結んだ国際調達会社（IPO）が行う．
- 品質保証として，出荷する全枚数に「オープンショート検査」を無料で実施．

■（株）インフロー
価格：¥43,000～
TEL：03-5228-7871　FAX：03-5228-7872
E-mail：info@p-ban.com
URL：http://www.p-ban.com/

●Web高速化ソフトウェア

FlyingServ Webキャッシュ

- ネットワーク回線への投資を抑えながら，Webシステムの応答性能を飛躍的に改善．
- Webコンテンツデータのハッシュ値を利用した差分管理機能により，動的コンテンツを効率よくキャッシュすることが可能．
- 静的コンテンツに対応する通常のキャッシュ機能，クライアントに流れるデータを圧縮して送る機能をサポートし，ネットワークを流れるデータの削減を実現．
- Webコンテンツの一部だけが変更されるような定型業務に対しては，データを大幅に削減することが可能となるため，低速なネットワーク環境で稼動するWebシステムでもコストの大きな回線投資をすることなく，少ない投資でWebシステムの応答速度を上げることが可能．
- Webサーバ側アプリケーションの変更はないため，既存のWebシステム環境に導入が容易．
- 検証支援サービスなどのサポートサービスも提供．

■（株）東芝
価格：¥1,500,000（サーバプロキシ/1CPU）
¥150,000（クライアントプロキシ/1CPU）
TEL：03-3457-2725

●開発ソリューション

CodeWarrior Development Studio for Symbian OS v2 OEM Edition

- JTAGハードウェアを使用してSymbian OS Ver.7のストップモードでのカーネルレベルデバッグを行える．
- UIQとSeries60をベースとする携帯電話用Symbianアプリケーション開発のための総合的なツール群も提供し，ROMに常駐するSymbianのアプリケーションフレームワークコードのデバッグが可能．
- MetroTRKも含まれており，リファレンスプラットホームとSymbian開発キットにより，ターゲットデバイス上でJTAG接続なしでシリアルケーブルを使ったデバッグを行うことが可能．
- グラフィカルなデバッガは，GNUデバッグツールと比較してさまざまな機能向上がなされている．

■ メトロワークス（株）
価格：下記へ問い合わせ
TEL：03-3780-6091　FAX：03-3780-6092

●Linuxカーネル/統合クロス開発環境

uLinux/ELITE

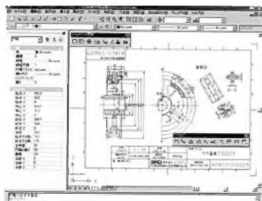
- uLinuxは同社が提供するLinuxカーネルであり，ELITEは，製品にLinuxベースの組込みソリューションをコンフィグレート，カスタマイズ，デプロイするための組み込みLinux開発環境.
- ELITEは，Linuxホストでの開発環境だけでなく，WindowsホストでもLinuxと同等の開発環境を提供.
- コンポーネントをモジュール化し，再利用や管理を促進するためのJavaベースのIDEを装備.
- プロジェクト生成ウィザード，コンフィグレーションツール，ビルドツール，ターゲットイメージエディタ，デプロイウィザード，デバッグウィザードの六つのツールにより構成.
- SH系，ARM系，MIPS系，PPC系およびMMUのないCPUに対応.
- リアルタイム機能，オプションでサードパーティのデバッガツールへの対応など，従来の開発ツールとは異なる特徴をもち，サードパーティを含むオプションソフトウェアスタックをスナップイン対応する機能も装備.

■ リネオソリューションズ（株）
価格：下記へ問い合わせ
TEL：03-5322-2856　FAX：03-5322-2858

●機械設計支援ソフト

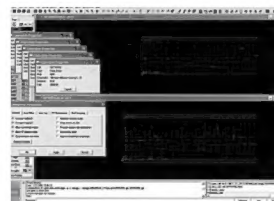
AutoMECH 2004

- オートデスク社のAutoCAD 2004に対応した機械設計支援ソフト.
- AutoCAD 2004に搭載されたVeriSignによる「ディジタル署名」と連動するハンコ機能を搭載．図面内にハンコ図形を挿入することで，図面ファイルの認証を行い，許可なく改ざんされていないことを保証.
- 部品表と表題欄の機能が一新され，入力されたデータはPDIを介してCSV/XMLに出力可能.
- 機械設計において多く利用される，寸法線機能を強化.
- 初期設定，補助線，JIS基本機械要素/地図記号の自動記入，異尺作図，作表，コネクタなど機械設計に必須の機能を多数搭載.

■（株）エス．アール．ディー
価格：¥248,000
TEL：06-6392-9511　FAX：06-6392-9524

●レイアウト2次元最適化ツール

Qtrek

- 米国Qデザイン社が開発した，ディープサブミクロンプロセスの半導体向けレイアウト2次元最適化ツール.
- 「Qtrek-Create」，「Qtrek-Migrate」の二つの製品で構成され，GDSフォーマットを経由したスタンドアロンで使用可能.
- SIIのULSIデザインシステムNew-SXおよびSX-9000と統合して動作.
- Qtrek-Createは，New-SX/SX-9000の環境内で，マスクレイアウトの最適化を行うことができ，New-SX/SX-9000のメニューから直接起動することが可能．データベースを統合しているため，マスクレイアウトの2次元最適化の実行結果をNew-SX/SX-9000で直接確認することができる.

■ セイコーインスツルメンツ（株）
価格：¥6,500,000～
TEL：03-5645-1741

●.NET用難読化ツール

Dotfuscator

- 米国プリエンプティブソリューションズ社が開発した，.NET開発環境向けの難読化ツール.
- 短い名前に置き換えるだけでなく，拡張されたオーバロードインダクション機能により，メソッドの戻り値の型を判定し，短い名前を重複利用するため，逆コンパイルによるソースの復元を最大限に困難にする.
- 文字列の暗号化を行う.
- コントロールフローに対して難読化が可能.
- サテライトDLLのシームレスな難読化が可能.
- 印字不能文字を対象とするリネームが可能.
- 不要なタイプ，メソッド，フィールドの除去や不要なコンスタントやインスタンス変数の除去も行い，最大70%までのサイズ縮小を実現.
- プログラムサイズの縮小にともないローディングに要する時間が短縮され，起動速度が向上.
- ILDASM対策機能や，XMLベースの使いやすい設定ファイルを用意.

■（株）エージーテック
価格：¥290,000
TEL：03-3293-5283　FAX：03-3293-5270
E-mail：info@agtech.co.jp
URL：http://www.agtech.co.jp/

●統合設計環境ツール

Qsim System

- ディジタル通信システムや信号処理向けに，アルゴリズム設計からRTL設計までをシームレスに設計する環境を提供する機能を多数搭載している.
- 単相同期方式とゼロ遅延モデルのシミュレーションに適したサイクルベース方式を採用したシミュレータにより，イベントドリブン方式と比較して高速なシミュレーションを実現.
- シミュレーション結果をロジック波形オシロスコープなどにより動作を確認でき，モデル中の任意の信号線をプローブを当てるような感覚で波形を見ることができる.
- ツールは動的にシミュレータに組み込むことが可能なプラグイン方式を採用しており，公開されているAPIを利用して必要な表示ツールを構築することも可能.
- MATLAB/Simulinkで記述したモデルをIP generatorツールにより自動的にRTLモデルに変換し，シミュレーションすることが可能.

■（株）キューウエーブ
価格：下記へ問い合わせ
TEL：03-3485-2900　FAX：03-3485-2900

●ディジタル文書作成ツール

Acrobat 6.0 Professional
Acrobat 6.0 Standard

- Acrobat 6.0 Standardは，ビジネスユーザー向けにPDFファイルの作成，レビュー，文書のやり取りのための使いやすいツールを提供.
- Microsoft OfficeのWord，Excel，Power Point，Outlookからボタン一つでPDFを作成．Windows版IEにもボタンを埋め込み，すべてのリンクを保持した状態で，ブラウザ内から自動的にWebページを保存.
- レビュートラッカーによって，レビューメンバのリストを自動的に作成し，送付した文書や注釈に対するコメントが返信されたかのトラッキングが可能．レビューは電子メールまたはWebブラウザを介して行うことができる.
- Acrobat 6.0 Professionalは，Standard版のすべての機能に加えて，AutoCAD，Visio，Projectなどの専門分野のアプリケーションからのPDF作成をサポート.
- AutoCADとVisioで作成されたPDFでは，レイヤ情報も保持．拡張されたPDF/X準拠とPostScriptレベルの互換性の検証が可能.

■ アドビシステムズ（株）
価格：¥54,800（Professional）
¥34,800（Standard）
TEL：03-5350-0407

海外・国内イベント/セミナー情報

INFORMATION

海外イベント

5/25-28　**International Symposium on Cirsuits and Systems**
Imperial Queen's Park Hotel, Bangkok, Thailand
IEEE
http://www.iscas2003.org/

6/2-6　**COMPUTEX TAIPEI**
Taipei World Trade Center, Taipei, Taiwan
CETRA
http://www.taipeitradeshows.com.tw/computex/

6/3-5　**Infosecurity Canada**
Sheron Centre Hotel, Toronto, Ontario, Canada
Reed Exhibitions
http://reedexpo.ca/infosec/

6/9-11　**NEPCON East/Electro**
Bayside Expo & Conference Center, Boston, MA, USA
Reed Exhibition
http://www.nepconeast.com/

6/17-19　**International Conference on Consumer Electronics**
Lax Marriott Hotel, Los Angels, CA, USA
IEEE
http://www.icce.org/

7/14-16　**SEMICON West 2003**
Moscone Center, San Francisco, CA, USA
SEMI
http://events.semi.org/semiconwest/

7/30-31　**Embedded Systems Conference Asia**
Taipei International Convention Center, Taipei, Taiwan
CMP Media Inc.
http://esconline.com/asia/

国内イベント

5/26-30　**INFORMATION STORAGE WEEK 2003**
東京ファッションタウン(TFT)ビル(東京都江東区)
国際ディスクドライブ協会 IDEMA Japan
http://www.idema.gr.jp/isw2003.htm

6/3-6　**RSA Conference 2003**
東京国際フォーラム(東京都千代田区)
Key3Media
http://www.key3media.co.jp/rsa2003/

6/4-6　**JPCA Show 2003(国際電子回路工業展)**
東京国際展示場(東京ビッグサイト，東京都江東区)
(社)日本プリント回路工業会
http://www.jpca.jp/2003/index.html

6/4-6　**ビジネスショウ OSAKA 2003**
インテックス大阪(大阪府)
(社)日本経営協会
http://www.noma.or.jp/bsosaka/

6/11-13　**画像センシング展**
パシフィコ横浜(神奈川県横浜市)
精機通信社
http://www.seiki-tsushin.com/sensing/

6/25-27　**設計・製造ソリューション展**
東京国際展示場(東京ビッグサイト，東京都江東区)
リードエグジビションジャパン
http://web.reedexpo.co.jp/dms/

7/9-11　**組込みシステム開発技術展 ESEC**
東京国際展示場(東京ビッグサイト，東京都江東区)
リードエグジビションジャパン
http://web.reedexpo.co.jp/ESEC/jp/

開催日，イベント名，開催地，問い合わせ先の順

セミナー情報

WindowsCE プリントシステムセミナー
開催日時　：5月27日(火)
開催場所　：みなとみらい 2-3-3 クイーンズタワー B 7F クイーンズフォーラム会議室(横浜市西区)
受講料　　：無料
問い合わせ先：(株)グレープシステム基本ソフトウェア事業部セミナー係，☎(045)222-3761，FAX(045)222-3759
http://www.grape.co.jp/seminar.html

パソコン・リアルタイム OS プログラミング技法
開催日時　：5月27日(火)～29日(木)
開催場所　：高度ポリテクセンター(千葉県千葉市)
受講料　　：35,000円
問い合わせ先：雇用・能力開発機構 高度ポリテクセンター事業課，☎(043)296-2582　http://www.apc.ehdo.go.jp/

わかりやすい情報通信ネットワーク
開催日時　：5月29日(木)
開催場所　：NTT アドバンステクノロジ会議室(東京都)
受講料　　：19,000円
問い合わせ先：サイペック(株)，info@r-sipec.jp
http://www.rlz.co.jp/seminar/seminardata.php?id=RGS305802

UWB(Ultra Wide Band)の基盤技術と規格動向
開催日時　：6月11日(水)～12日(木)
開催場所　：中央大学駿河台記念館(東京都千代田区)
受講料　　：68,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c030611a.htm

Linux デバイスドライバプログラミング
開催日時　：6月11日(水)～13日(金)
開催場所　：高度ポリテクセンター(千葉県千葉市)
受講料　　：30,000円
問い合わせ先：雇用・能力開発機構 高度ポリテクセンター事業課，☎(043)296-2582

UML による組み込みシステムモデリング入門(演習つき)
開催日時　：6月12日(木)
開催場所　：CQ 出版セミナールーム
受講料　　：18,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

PC 実習!! TCP/IP ネットワーク・プログラミング
開催日時　：6月12日(木)～13日(金)
開催場所　：SRC セミナールーム(東京都高田馬場)
受講料　　：78,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_143.htm

C 言語ベースのシステム LSI 設計
開催日時　：6月13日(金)
開催場所　：CQ 出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

組み込み用ボードを使った Linux システム設計入門
開催日時　：6月16日(月)
開催場所　：アドバンスト・テクノロジーセンター(東京都千代田区)
受講料　　：73,500円
問い合わせ先：(株)アドバンスト・テクノロジーセンター，☎(03)3518-6441，FAX(03)3518-6147
http://www.at-center.co.jp/pdf/A_1279.pdf

リアルタイム OS の基礎
開催日時　：6月19日(木)
開催場所　：CQ 出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

スミスチャートを使った高周波回路設計の基礎
開催日時　：6月20日(金)
開催場所　：CQ 出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

ATA(IDE)/ATAPI 技術解説
開催日時　：6月26日(木)～27日(金)
開催場所　：中央大学駿河台記念館(東京都千代田区)
受講料　　：68,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c030626a1.htm

Windows Server 2003 におけるシステムへの展開・運用・管理
開催日時　：6月26日(木)～27日(金)
開催場所　：SRC セミナールーム(東京都高田馬場)
受講料　　：76,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_145.htm

XPort ソリューションセミナー
開催日時　：6月27日(金)
開催場所　：日新電機東京支社
受講料　　：55,000円
問い合わせ先：(株)日新システムズ，☎(03)5807-5931，FAX(03)3839-0112
http://www.co-nss.co.jp/index.html

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

読者の広場

アンケートの結果

興味のあった記事（2003年5月号で実施）

①プロローグ 組み込み機器開発における問題点と解決手法
②第1章 組み込みソフトの開発を考えたとき不足していること
③第4章 組み込みソフト開発にオブジェクト指向を導入した成果の測定手法
④第2章 エレベータモデルの検証
⑤第8章 開発効率化のための組織とAgile方法論
⑥第5章 どうして組み込み分野でオブジェクト指向が広まらないのか
⑦フリーソフトウェア徹底活用講座（第9回）
⑧第6章 事業としての組み込み機器開発の課題を整理する
⑨第3章 電波時計システムモデルの検証
⑩第9章 人的協働モデルとモデル作成者の適性
⑪第7章 ソフトウェアプロダクトラインとプロセスの定義
⑫音楽配信技術の最新動向（第4回）
⑬NET&COM 2003
⑭開発技術者のためのアセンブラ入門（第18回）
⑮InterGiga No.30
⑯組み込みプログラミングノウハウ入門（第11回）
⑰Microwindowsを使った組み込み向けGUIプログラムの作成事例（基礎編）
⑱エピローグ 組み込み機器開発効率化のための今後の取り組み
⑲ハッカーの常識的見聞録（第29回）
⑳シニアエンジニアの技術草子（弐拾七之段）
㉑プログラミングの要（第3回）

特集『うまくいく！組み込み機器の開発手法』についてのアンケートの結果

Q1 今回のように，現場エンジニアとコンサルタントといった，異なる立場からの解説で構成される特集は？

①非常におもしろくたいへん興味がある（45%）
②仕事上でも十分に参考になる（27%）
③学習/評価用に役立つ（18%）
④話としてはおもしろい（9%）
⑤役に立たない（0%）
⑥その他（0%）

Q2 今月号の特集はいかがでしたか？

①よく理解できた（36%）
②ある程度理解できた（63%）
③あまり理解できなかった（0%）
④ほとんど理解できなかった（0%）
⑤関係ない分野だった（0%）
⑥その他（0%）

Q3 今後，開発手法/オブジェクト指向関連の記事に期待する内容/テーマ/トピックは？（複数回答可）

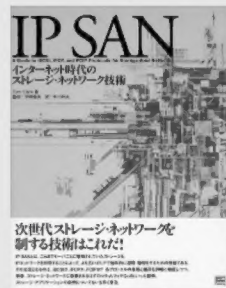
①オブジェクト指向プログラミング（5%）
②オブジェクト指向方法論（5%）
③UML記法の解説（5%）
④UMLの適用事例（12%）
⑤プロジェクトメトリックス（3%）
⑥Rational Unified Process（0%）
⑦オブジェクト指向分析（10%）
⑧オブジェクト指向設計（8%）
⑨デザインパターン（5%）
⑩組み込み向けオブジェクト指向（15%）
⑪eXtream Programming（8%）
⑫Agile方法論（15%）
⑬CMM（3%）
⑭プロダクトライン（5%）
⑮その他（0%）

特集担当デスクから

☆最近のPC/AT互換機では，最新Pentium4の800MHzFSBなど，一昔前では考えられなかった周波数で動いています．今回の特集で，そのからくりを理解していただけたかと思います．
☆バスは一般的に，性能を追求すれば互換性が問題になり，互換性を重視すれば高性能化が難しくなります．規格化におけるそのバランスは，非常に重要な点です．USB2.0のハイスピードデバイスは，互換性を確保しつつ大幅な性能向上を実現できたわけですが，その内部は互換性のための回路を含むなど，複雑になっているのがわかります．

読者プレゼント

●応募方法：本誌読者アンケートはがきに必要事項を記入のうえ，2003年6月30日（必着）までにご応募ください．なお当選者の発表は，発送をもってかえさせていただきます．

（1）『IP SAN
　ーインターネット時代の
　ストレージ・ネットワーク技術』（1名）

次号予告 現代コンピュータ技術の基礎

2003年8月号は6月25日発売です

Linux/BSD/シミュレーション/データベース/ARM/文字コード処理/コンピュータシステム/ワイヤレスネットワーク/ICカード/USB

最近のコンピュータ/エレクトロニクス技術は進歩がとても速く，新しい技術が次々に開発されてくるうえに，細分化も進み，技術の動向を追うだけでもたいへんである．開発現場のエンジニアは，自分の専門以外の分野だと，内容をなかなか理解できないこともある．そこで次号特集では，最近の本誌の特集記事(2002年7月号～2003年4月号)の中に出てきた，さまざまな用語について，詳細な技術解説を行う．本誌特集は，その時点における重要技術を掘り下げて解説しており，その用語を理解することは，ひいてはコンピュータ/エレクトロニクス技術の流れを理解する「早道」ともなる．

また，別冊付録として，開発現場で役立つ「数式・公式集」が付属する．

編集後記

■新緑に浸って気分転換！とばかりに，自宅から10個ほど「奥」の駅へ家族で降り立ち，軽い山登り＆川遊びをしてきた．開けた景色で飲む缶Beerはとっても美味だった．翌日，痛みを訴える足に，「やった，まだ若い！」と家人に「申告」したら，「私は痛くない．その程度で痛くなるとは足腰弱いねー」と「逆襲」されてしまった．(洋)

■とあるブラジリアン柔術の道場に次のような張り紙がしてある．「タップは恥ずかしいことではありません」．タップとは，関節を極められた場合に，「まいった」の意思表示をすることである．確かに，タップもせずに無理して我慢したら，一生何もできない体になりかねない．「タップ」は決して恥ずかしいことではない．(＝IO)

■ゴールデンウィークに編集作業の山場を迎えるうちの編集部にとって，今年はぜんぜん“ゴールデン”じゃなかったので作業も滞りなく進み……っというのは甘くて，筆者校正が大変．電話番号から察するに，旅行先？にFAXを送ることになった筆者や，某筆者は巣鴨で缶詰に(!!)お休みのところ申し訳ありませんでした．(M)

■ノートPCのバッテリが死亡．まったく使えなくなったことから，最初は接触不良を疑ったのですが，リチウムイオン電池は劣化すると，出力が0Vになるのだそうです．ニッカドだったらいくら劣化しても少しは使えるのですが……．正しい知識を身に付けていれば，時間を浪費することもなかったのですが．(み)

■バブルが崩壊してずいぶん経つが，イラク戦争に伝染病と経済にとって悪いことが続く．失業率も上がっているしリストラ対象になっている人も多いが，不思議と日本経済が深刻な状況になっているようには感じられない．鈍感になっているのだろうか，自分の感覚が少し心配だ．しかし，良い話もなさそうなので，もっと酷い状況を覚悟しなければならないのかも．(Y)

■相方の友達が戸建を買ったので，御祝いに友達数人でスロットの実機を贈ったそうです．夫婦揃ってスロ好きで，欲しいと思ってもなかなか買えない物なので，たいへん喜ばれたとか．贈り物する時ってけっこう悩むので喜ばれて幸いですが，さすがにこれほどインパクトのある贈り物ってそうそうないのでは(笑)．(Y2)

■今年に入ってから，山手線の外側を利用した新たな路線の繋がりが頻繁に告知されており，起点から目的地への最短の距離・時間・費用の検索に情報提供各社が追いついていかないようだ．特に，展示会場がある最寄り駅への行き方は何通りも出来ており，利用する路線の選択・組み合わせを考えさせられる今日この頃．(S)

■姉に赤ちゃんができたことがわかり，我が家では連日その話題でもちきりらしい．特に初孫とあってか母が大騒ぎしている．生まれる前でこうなのだから，生まれたらどうなることやら……．姉より母の方が心配である．私も楽しみにしている一人ではあるが．(8)

お知らせ

▶読者の広場
本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

▶投稿歓迎
本誌に投稿をご希望の方は，連絡先(自宅/勤務先)を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です(送り先はsupportinter@cqpub.co.jpまで)．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

▶本誌掲載記事についてのご注意

▶コピーサービスのご案内
本誌バックナンバーの掲載記事については，在庫(原則として24か月分)のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．
- コピー料金(税込み)
  1ページにつき100円
- 発送手数料(判型に関わらず)
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461　東京都豊島区巣鴨1-14-2
  CQ出版株式会社　コピーサービス係
  (TEL：03-5395-4211，FAX：03-5395-1642)

▶お問い合わせ先のご案内
- 在庫，バックナンバー，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface
©CQ出版(株)　2003　振替　00100-7-10665
2003年7月号　第29巻　第7号(通巻第313号)
2003年7月1日発行(毎月1日発行)
定価は裏表紙に表示してあります

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1－14－2
電話／編集部(03)5395－2122　URL　http://www.cqpub.co.jp/interface/
広告部(03)5395－2133　インターフェース編集部へのメール
販売部(03)5395－2141　supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1－14－2 Sugamo，Toshima-ku，Tokyo 170-8461，Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社

発行人／蒲生良治
編集人／相原　洋
編集／大野典宏　村上真紀　山口光樹　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／神崎真理子　森　祐子　唐沢睦子
広告／澤辺　彰　中元正夫　菅原利江

日本ABC協会加盟誌
(新聞雑誌部数公査機構)　ISSN0387-9569

Printed in Japan